UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
DEPARTAMENTO DE TELEMÁTICA

PROYECTO FIN DE CARRERA
INGENIERÍA DE TELECOMUNICACIÓN

GESTIÓN DE RIESGO EN DISPOSITIVOS ANDROID BASADA EN ELIMINACIÓN DE VULNERABILIDADES Y DETECCIÓN DE CONTEXTOS

Autora: Laura de Matías García

Directora: Florina Almenares Mendoza

Tutora: Patricia Arias Cabarcos

Leganés, 2013

Título: GESTIÓN DE RIESGO EN DISPOSITIVOS ANDROID BASADA EN ELIMINACIÓN DE VULNERABILIDADES Y DETECCIÓN DE CONTEXTOS

Autora: Laura de Matías García

Directora: Florina Almenares Mendoza

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Un viaje de diez mil kilómetros empieza por un solo paso.

Este concluye aquí, dando pie a nuevos viajes.

Agradecimientos

Me gustaría mostrar mi agradecimiento a todas aquellas personas que de un modo u otro han contribuido a hacer posible el haber llegado hasta este momento.

En primer lugar, quisiera agradecer a mi tutora y a mi directora del proyecto, Patricia Arias Cabarcos y Florina Almenares Mendoza por su disponibilidad, ayuda y profesionalidad a lo largo de la realización de este proyecto.

A mis padres y hermana por su apoyo incondicional durante todos estos años. Gracias por vuestra confianza y comprensión, y por recordarme la importancia del trabajo bien hecho, la constancia, y la superación.

A David, por haberme acompañado a lo largo de este camino. Gracias por haber estado siempre ahí, por toda tu ayuda y por no haber dejado nunca de creer en mí.

Por último agradecer también a mis compañeros de clase y amigos, tanto a los que me han acompañado desde el inicio de esta etapa como los que se han ido incorporando a lo largo de ella, por su inestimable ayuda y los buenos momentos vividos, los cuales han contribuido a afrontar el día a día con optimismo.

A todos, gracias.

Resumen

En la actualidad, los *smartphones* se han convertido, en poco tiempo, en los dispositivos de comunicación más utilizados. Las diversas funcionalidades que ofrecen estos terminales implican la exposición y el acceso a una gran cantidad de información personal y confidencial por parte de las aplicaciones instaladas en ellos.

Android se trata del sistema operativo móvil más utilizado. Sin embargo, al tratarse de un sistema joven, no cuenta aún con suficientes mecanismos para la mitigación del riesgo presente en él y sus aplicaciones. Asimismo, su configuración de seguridad se trata de una labor tediosa que conlleva la falta de implicación por parte del usuario.

Este proyecto tiene como objetivo proporcionar un mayor control sobre los riesgos de seguridad en Android. Más concretamente, se pretende incrementar el conocimiento y el control sobre las posibles vulnerabilidades presentes en las aplicaciones, además de contribuir a la adaptabilidad automática de la seguridad del dispositivo en función de su entorno.

El sistema implementado cuenta con un gestor de vulnerabilidades y un módulo de seguridad por contexto e interactúa con la NVD, repositorio público de vulnerabilidades software estadounidense y la API de Android, tras la evaluación de varias alternativas como posibles fuentes de información.

En definitiva, se presenta el desarrollo de una aplicación que, basada completamente en herramientas libres de desarrollo, logra mitigar el riesgo presente en *smartphones* con sistema operativo Android y se sientan unas bases para, a partir de esta aplicación, continuar con la investigación y mejora de la seguridad en él.

Abstract

In only a few years, smartphones have become one of the most commonly used communication devices due to their versatility. But all that different functionalities imply the access to a lot of personal and confidential information from the installed applications.

Nowadays, Android is one of the most important mobile operative systems. However, due to its youth, there is a lack of risk mitigation mechanisms. Besides, its security configuration is tedious, which involves no implication of the user.

This project pretends to provide a better control over security risks in Android. Its main is to increase the knowledge and control over possible vulnerabilities in applications and contribute to an automatic adaptable security device configuration depending on the environment the smartphone is in.

The implemented system counts with a vulnerability manager and a context based security module, and it interacts with the NVD, U.S. government repository of software vulnerability data, and the Android's API, after the evaluation of other alternatives as information sources.

In conclusion, it is introduced an application that, based on free software development tools, contributes to mitigate risks in Android and that lays the foundations of new ideas for the security investigation on this operative system.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1. Motivación

Cada vez más se confía en los *smartphones* para almacenar información personal y para acceder a todo tipo de servicios *online* a través de ellos. En este mundo móvil, el sistema operativo Android es actualmente el que tiene una mayor penetración en el mercado y el que goza de un mayor número de usuarios. Es por ello que los dispositivos móviles con dicho sistema operativo se han convertido en objetivo principal de ataques de seguridad. De hecho, el 79% de los ataques a móviles llevados a cabo el año pasado se realizaron contra terminales Android [1].

Por las razones anteriormente expuestas, disponer de herramientas que permitan analizar el riesgo es imprescindible para mejorar la seguridad. El riesgo, generalmente expresado de forma matemática como $R = P \times I$, es proporcional tanto a la probabilidad de ocurrencia de un evento negativo como al impacto que dicho evento pueda tener. Además, una vez conocido el riesgo, se pueden poner en marcha salvaguardas; es decir, tomar acciones que permitan mitigarlo o eliminarlo.

De acuerdo con los conceptos explicados, la motivación principal en este proyecto fin de carrera es obtener aquellos factores que contribuyen en un dispositivo Android a aumentar la componente de probabilidad (P)[1]. Así se tiene en todo momento una noción del riesgo existente y, por ende, de la susceptibilidad de sufrir ataques. Además, esta información puede ser utilizada como base para tomar decisiones de seguridad que mitiguen o eliminen el riesgo de ataques. Para demostrar esto, se programa también una aplicación que permite: a) reducir el riesgo relacionado con vulnerabilidades *software*; y b) adaptar automáticamente el nivel de seguridad cambiando el mecanismo de control de acceso al terminal dependiendo del riesgo asociado al contexto en el que se encuentre el usuario.

1.2. Objetivos

El objetivo de este proyecto es proporcionar un mayor conocimiento y control sobre los riesgos presentes en la seguridad de los dispositivos móviles Android. Estos riesgos pueden referirse a vulnerabilidades de seguridad en las aplicaciones instaladas o al nivel de inseguridad de un determinado entorno, en el que puede encontrarse el terminal.

[1] El impacto está sujeto al tipo de información almacenada y tratada en el *smartphone*, y su evaluación depende de la visión subjetiva del usuario

Asimismo se pretende añadir adaptabilidad en la configuración de seguridad a través de la modificación automática de algún parámetro o método de seguridad de los dispositivos.

Siguiendo estos fines, se implementará un gestor de vulnerabilidades, encargado de informar al usuario sobre las vulnerabilidades presentes en el sistema y sobre sus posibles soluciones. Dicho gestor incluye, por una parte el cliente móvil y, por otra parte, un servidor de datos. Además, se desarrollará un módulo de seguridad por contexto para la detección de entornos y la adaptabilidad de la configuración de seguridad del terminal en función de dichos entornos. La aplicación estará disponible para las versiones comprendidas entre 2.2 y 4.2 de Android y utilizará para su desarrollo solamente herramientas libres.

1.3. Estructura del documento

Este documento se divide en siete capítulos, cuya función se expone a continuación:

El presente Capítulo 1, "Introducción", recoge la motivación para la realización de este proyecto, sus objetivos y la estructura de esta memoria.

En el Capítulo 2, "Estado del arte", se exponen las tecnologías utilizadas a lo largo de este proyecto. Para ello, se divide su estudio en tecnologías base y tecnologías relacionadas con la gestión del riesgo. Dentro de las tecnologías base, se realiza una explicación detallada del sistema operativo móvil Android y del concepto de seguridad por contexto, y se presentan los objetos Servlet y el contenedor de Servlets Tomcat, utilizados en el desarrollo del servidor. En cuanto a las tecnologías relacionadas con la gestión del riesgo, se explica la NVD y todos sus recursos relacionados.

El Capítulo 3, "Descripción general del sistema", trata sobre el diseño del sistema a desarrollar, con la explicación de su arquitectura, funcionalidad y requisitos. Asimismo, se presentan los módulos que componen la aplicación.

En el Capítulo 4, "Implementación del sistema", se recoge el desarrollo de la aplicación, junto con sus clases y métodos, y el cual se divide en la interacción con el usuario y la lógica interna. En la interacción con el usuario, se expone la implementación y las vistas que componen la interfaz. Posteriormente, en la lógica interna se explica el desarrollo de los módulos que dan funcionalidad al sistema.

En el Capítulo 5, “Pruebas”, se describen las pruebas llevadas a cabo en cada módulo de la aplicación y su resultado. Además, se realiza la comprobación de que estas pruebas engloban todos los requisitos funcionales del sistema.

El Capítulo 6, “Histórico del proyecto”, presenta una descripción de las distintas fases para el desarrollo de este proyecto, explicando las tareas, problemas y resultados obtenidos en cada una de ellas y un resumen con su planificación temporal.

Por último, el Capítulo 7, “Conclusiones y líneas futuras”, recoge las conclusiones obtenidas tras la finalización del proyecto y se presentan futuras líneas de trabajo para su ampliación.

Además de estos siete capítulos, al final del documento se incluyen cinco anexos que completan algunas partes de la explicación este proyecto:

En el Anexo A, “Presupuesto”, se presenta el presupuesto del proyecto.

En el Anexo B, “Entorno de trabajo”, se describen las diferentes herramientas utilizadas para la implementación del proyecto.

El Anexo C, “Manual de usuario”, se trata de una guía en la que se explican los pasos para el uso de la aplicación.

El Anexo D, “NIDS para Android”, recoge la descripción de varios posibles sistemas de detección de intrusos de red para el sistema operativo Android.

Para finalizar, en el Anexo E, “Glosario”, se incluye un glosario de términos.

Capítulo 2. Estado del arte

En este segundo capítulo se presentan las tecnologías y conceptos que han sido analizados y utilizados para el desarrollo de este proyecto.

El primero de sus dos bloques está dedicado a las tecnologías base empleadas. En él, se presenta la plataforma Android, sistema operativo sobre el que se desarrolla la aplicación. Se describen sus características, su arquitectura y la de sus aplicaciones y se explican los fundamentos de la seguridad en Android en la actualidad. Tras ello, se introduce la tecnología Servlet, también empleada en este proyecto a través de la utilización del contenedor de Servlets Tomcat.

En el segundo de los bloques se da a conocer una tecnología relacionada con la gestión del riesgo, la base de datos NVD, de la que se explican los diferentes recursos en los que se basa su contenido, los cuales son utilizados en la aplicación.

2.1. Tecnologías base

2.1.1. Android

2.1.1.1. Introducción

Android [2] es una plataforma operativa móvil propiedad de Google y supervisada por el OHA (Open Handset Alliance). Basada en el núcleo de Linux, se trata de un conjunto completo de *software* para dispositivos móviles que agrupa un sistema operativo, *middleware* y aplicaciones móviles básicas. Esta estructura permite abstraer al sistema operativo de las aplicaciones a través del *middleware* basado en la máquina virtual Dalvik.

Android dispone de toda una plataforma de código abierto para el desarrollo de aplicaciones. Google proporciona el SDK (Software Development Kit) de Android que provee a los programadores de las herramientas y APIs (Application Programming Interface) necesarios para el desarrollo de aplicaciones mediante el uso del lenguaje de programación Java. Asimismo, el SDK permite desarrollar aplicaciones para cualquiera de las versiones de Android, las cuales son compatibles hacia delante.

2.1.1.2. Historia

Los orígenes de Android se sitúan en 2007, año en el que se crea el OHA. El OHA [1] es un consorcio de 48 empresas tecnológicas cuyo fin es el desarrollo de estándares abiertos para terminales móviles y que centra sus esfuerzos en la difusión de la plataforma Android. Algunos miembros destacados de esta alianza son Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson o Vodafone.

El 5 de noviembre de 2007 Google junto a OHA lanzó una primera versión del SDK de Android y al año siguiente apareció el primer móvil con este sistema operativo, el 'T-Mobile G1' de la compañía T-Mobile. En octubre de 2008, Google libera el código fuente de Android bajo licencia de código abierto Apache, siguiendo así con la filosofía de *software* libre de la OHA por la que se posibilita a desarrolladores y usuarios la creación de nuevas aplicaciones y la introducción de mejoras en las ya existentes. Ese mismo año se creó también el Android Market para la subida y descarga de aplicaciones.

Desde 2009 hasta la actualidad, Google ha continuado lanzando sucesivas versiones del SDK en las que se han ido optimizando e incorporando nuevas características, así como facilitando las herramientas necesarias para la implementación de programas propios [3]. Asimismo se han desarrollado versiones exclusivas para *tablets* como la 3.2 o la 4.0 para móviles y *tablets*, siendo la versión 4.3 la más reciente.

Hasta la actualidad, Android se ha ido consolidando como uno de los sistemas operativos móviles más utilizados y mantiene su tendencia creciente (Figura 1). A ello han contribuido diversos factores entre los que se encuentran su correcto funcionamiento en una amplia variedad de dispositivos *hardware* y la inexistencia de costes relacionados con el pago de licencias de *software* lo cual propicia bajos costes de integración y desarrollo.

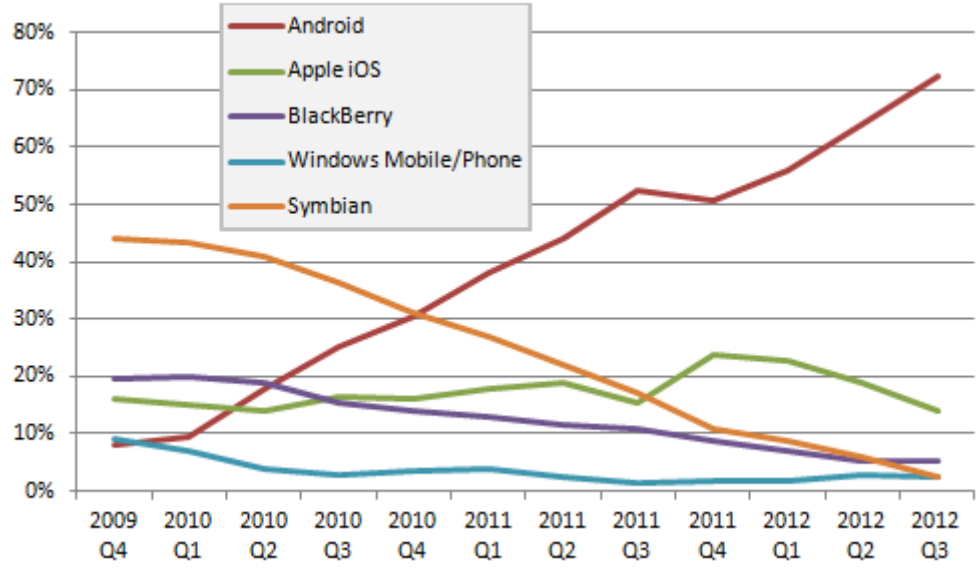

Figura 1. Cuota de mercado en el mundo de los sistemas operativos móviles hasta el tercer cuarto de 2012 [4]

2.1.1.3. Características

Android ha sido el primer sistema operativo para móviles capaz de combinar potentes características para diferentes tipos de dispositivos. Seguidamente, se presentan algunas de las más destacadas [5].

La principal de sus características es que se trata de una plataforma de código abierto basada en Linux. Esto facilita el desarrollo de nuevas aplicaciones sin que sea necesario el pago de *royalties*. Asimismo, dispone de librerías para la reutilización de código así

como de un amplio entorno de desarrollo que incluye un emulador de los diferentes dispositivos móviles y herramientas de depuración.

Android es un sistema operativo adaptable a cualquier tipo de dispositivo *hardware*, permitiendo su uso en una gran variedad de sistemas, y sus aplicaciones están implementadas en Java lo que garantiza su ejecución en cualquier tipo de procesador, siendo la portabilidad una de sus características más importantes.

En relación a su arquitectura, ésta basa algunos de sus componentes en Internet, como ocurre con los ficheros XML (eXtensible Markup Language) para el diseño de las interfaces de las aplicaciones que facilitan que una misma aplicación pueda ser observada correctamente en diversas pantallas.

Otra de sus características es la optimización de recursos. Para ello Android dispone de su propia máquina virtual encargada de realizar una gestión eficiente de las aplicaciones y su memoria.

En cuanto a prestaciones multimedia, Android cuenta con una alta calidad de gráficos y sonido. Además proporciona soporte para los formatos de imagen y audio más habituales (JPG, PNG, GIF, MPEG4, MP3).

Otras de sus prestaciones destacables son que dispone de una base de datos conocida como SQLite que se integra de forma directa con las aplicaciones y de un navegador basado en el motor de código abierto WebKit al que se le pueden añadir más características. También cuenta con localización basada en GPS (Global Positioning System) y redes y reconocimiento y síntesis de voz.

2.1.1.4. Arquitectura

La arquitectura de Android [5] está compuesta de cuatro capas funcionales basadas en *software* libre: el núcleo de Linux, el runtime de Android, las librerías nativas, el entorno de aplicación y las aplicaciones. En la Figura 2 se muestra cada una de estas capas y sus principales componentes, las cuales se explican a continuación.

Núcleo Linux

El núcleo de Android emplea el sistema operativo Linux, concretamente su versión 2.6 y es el encargado de las funciones principales del sistema tales como la seguridad, el manejo de memoria, la gestión multiproceso, la pila de protocolos y el control de los drivers de dispositivos.

Asimismo, esta capa proporciona un nivel de abstracción entre la implementación *hardware* y el resto de capas *software* de la arquitectura de Android.

Runtime de Android

El entorno de ejecución está formado por las *Core Libraries*, que recogen las librerías con la mayoría de las funcionalidades de Java, aunque su componente principal es la máquina virtual Dalvik. La máquina virtual Dalvik, encargada de la ejecución de las aplicaciones, se trata de una versión de la máquina virtual de Java estándar pero optimizada para dispositivos móviles debido a las restricciones de procesamiento y memoria impuestas por estos. Igualmente, está diseñada para que un mismo dispositivo pueda ejecutar múltiples máquinas virtuales eficientemente.

Cada una de las aplicaciones en Android se ejecuta en su propio proceso y tiene su propia instancia de la máquina virtual Dalvik. Dicha máquina está basada en registros y utiliza ficheros Dalvik ejecutables (formato .dex) compilados y optimizados para el ahorro de memoria.

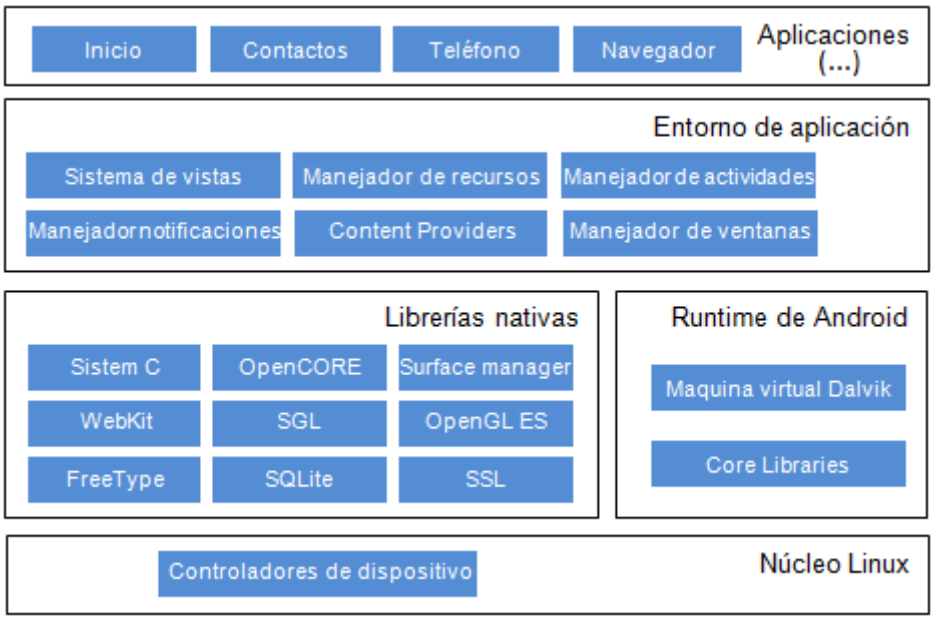

Figura 2. Arquitectura de Android [5]

Librerías nativas

Se trata de la capa compuesta por las bibliotecas o librerías nativas de Android cuya finalidad es la de facilitar funcionalidades a las aplicaciones mediante la utilización de código ya implementado. Están escritas en C o C++ y compiladas en función del código nativo de cada procesador.

Algunas de las librerías más destacadas son *System C Library*, basada en la librería BSD de C estándar (libc); *Media Framework*, versión basada en PacketVideo's OpenCORE y que da soporte a códecs de reproducción y grabación en una gran diversidad de formatos; *Surface Manager*, para el acceso al subsistema de representación gráfica en 2D y 3D; o SQLite, para el manejo ligero de bases de datos de las aplicaciones.

Entorno de aplicación

Es la capa encargada de proporcionar un entorno de desarrollo simple a través de la reutilización de componentes como sensores, servicios, o localización; y de las capacidades de otras aplicaciones, las cuales pueden hacerse públicas aunque con acceso restringido por seguridad.

Una característica importante del entorno de aplicación es que se aprovecha el lenguaje Java para todo aquello que el SDK de Android no es capaz de ofrecer en su estándar del JRE (Java Runtime Environment).

Los servicios destacados en este nivel son el sistema de vistas, para la interfaz visual de las aplicaciones; el manejador de recursos, que proporciona el acceso a los recursos que no son código; el manejador de actividades, encargado del ciclo de vida de las aplicaciones y de la navegación entre las mismas; el manejador de notificaciones, para las alertas en la barra de estado del dispositivo; y los proveedores de contenidos, que facilitan el acceso a los datos de otras aplicaciones.

Aplicaciones

Es el nivel formado por las aplicaciones instaladas en el dispositivo Android, todas las cuales corren en la máquina virtual Dalvik. Las aplicaciones se presentan en mayor detalle en el siguiente apartado.

En este nivel se encuentra además la aplicación Inicio, la cual permite la ejecución de otras aplicaciones desde los diferentes escritorios y la ejecución de *widgets*.

2.1.1.5. Aplicaciones

Las aplicaciones son programas generalmente escritos en Java y desarrollados mediante la utilización del SDK de Android. No obstante, también existe la posibilidad de su programación en C o C++ mediante el uso del NDK (Native Development Kit) de Android.

Las aplicaciones están formadas por componentes que se encargan de las diferentes funciones dentro de ella. Además, estos componentes siguen un ciclo de vida que establece un control sobre los mismos. Estos aspectos sobre las aplicaciones junto a otros de relevancia son explicados en este apartado.

2.1.1.5.1. Componentes de una aplicación

Las aplicaciones en Android se basan en una serie de bloques de construcción también llamados componentes a través de los cuales se define el comportamiento de la misma. Cada uno de estos componentes son entidades propias con un papel específico aunque se pueden comunicar ellos, creándose una dependencia entre los mismos. Esta comunicación se realiza mediante el uso de intenciones a través de las cuales es posible el paso de unos componentes a otros y el intercambio de información entre ellos.

Son cuatro las diferentes clases de componentes que una aplicación puede albergar: actividades, servicios, receptores de anuncios y proveedores de servicios. Seguidamente se explica la función y características [5] de cada una de ellas.

Actividad

En Android las aplicaciones están compuestas de una serie de elementos básicos de visualización conocidos como actividades. Las actividades son, por tanto, las diferentes pantallas que un usuario ve en el dispositivo y que permiten su interacción con él.

Existe una actividad principal, que será la primera en ejecutarse al lanzarse la aplicación. Después, el movimiento entre pantallas se realiza llamando a nuevas actividades. Así, cada actividad puede lanzar una nueva con el fin de llevar a cabo una acción diferente aunque todas tengan un objetivo común. El usuario podrá navegar hacia delante y hacia atrás entre estas pantallas ya que el sistema mantiene una pila de actividades a la que se van añadiendo las actividades empezadas.

Cada actividad actúa como interfaz de usuario a través del uso de vistas y *layouts*. La implementación de una actividad consiste en la creación de una subclase de la clase *Activity* junto con la definición de su interfaz mediante una jerarquía de vistas, es decir, objetos descendientes de la clase *View*, por lo que pueden definirse en código Java aunque lo más habitual es hacerlo en un fichero XML y que después sea el sistema el encargado de crear los objetos a partir del fichero. Un conjunto de vistas componen un *layout*, ayudando a su organización dentro de la pantalla de visualización.

Servicio

Un servicio es un proceso que puede ejecutarse sin que sea necesaria la interacción del usuario, por lo que no poseen interfaz gráfica. Existen dos clases de servicios en Android, los servicios locales, que pueden ser utilizados por aplicaciones que se encuentren dentro del mismo dispositivo y los servicios remotos, que pueden ser accedidos desde otros terminales.

Estos procesos en segundo plano se utilizan, por tanto, para procesos cuya ejecución debe ser continua esté o no visible o activa la interfaz de usuario. Asimismo, son útiles para ejecuciones de largo período de tiempo o en el caso de procesos de control remoto.

Receptor de anuncios

El receptor de anuncios es el encargado de la recepción y reacción frente a anuncios de tipo *broadcast* tanto si son originados por el sistema, en el caso del aviso de batería baja o de una llamada entrante, como si proceden de aplicaciones. Se trata de un código inactivo activado al recibirse el evento con el que está relacionado.

A pesar de no contar con interfaz de usuario pueden lanzar una actividad para atender a un anuncio. También pueden crear notificaciones a mostrar en la barra de estado del dispositivo.

Proveedor de contenido

En Android cada aplicación se ejecuta independientemente, de forma que sus datos quedan aislados del resto de aplicaciones del sistema. Sin embargo, en ocasiones se hace necesario poder disponer de los datos de otras aplicaciones. Para la compartición de esta información es para lo que se utilizan los proveedores de contenido.

Mediante este mecanismo se permite el intercambio de datos sin comprometer la seguridad del sistema de ficheros. Esto es así debido a que las aplicaciones pueden acceder o modificar datos de otra solamente en función de los permisos definidos para ello.

2.1.1.5.2. Ciclo de vida de una aplicación

El ciclo de vida de una aplicación [5] recoge las diferentes etapas que pueden seguir sus procesos desde que esta comienza hasta que finaliza.

Este ciclo no está controlado directamente por la aplicación, si no que es el sistema el encargado de ello. Las aplicaciones se ejecutan mediante sus propios procesos Linux, los cuales son controlados por el sistema Android en función de las necesidades del usuario y de los recursos disponibles. En memoria se mantienen los procesos el mayor tiempo posible, sin embargo, puede hacerse necesaria la eliminación de algunos de ellos. Con el fin de establecer qué procesos son los que se deben mantener en el sistema se define una jerarquía de procesos.

Son cinco los niveles en la jerarquía de importancia de los procesos de una aplicación en Android. Estos están representados de mayor a menor prioridad en la Figura 3 y se explican a continuación.

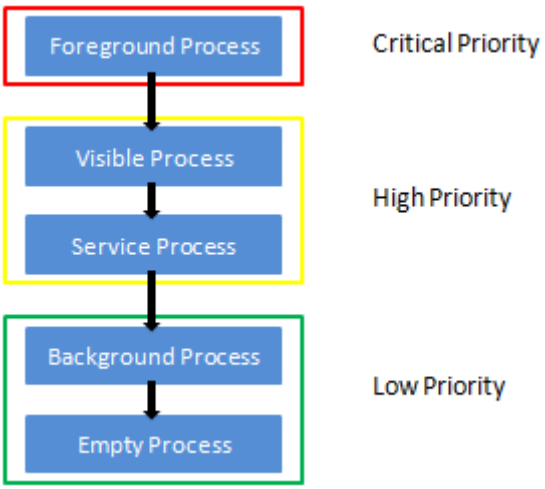

Figura 3. Jerarquía de los procesos de una aplicación en Android

Proceso de Primer Plano (*Foreground Process*)

Se trata de un proceso de primer plano para la acción que desarrolla el usuario en ese instante. Son pocos los procesos de este tipo ejecutándose al mismo tiempo en el sistema y solamente son eliminados en caso de que la memoria disponible sea baja. En cualquier otro caso no se detendrá su ejecución.

Proceso Visible (*Visible Process*)

Son procesos que aunque no disponen de ningún componente en primer plano, puede afectar a lo que el usuario se encuentre visualizando en pantalla. Debido a su importancia, no son eliminados a excepción de que sea necesario para mantener la ejecución de los *Foreground Process*.

Proceso de Servicio (*Service Process*)

Es el proceso para la ejecución de servicios por lo que realiza acciones requeridas por un usuario pero sin afectar a lo mostrado en pantalla. Se tratan de procesos de alta prioridad por lo que no deben ser eliminados mientras se disponga de memoria suficiente, no obstante, su importancia es menor que la de los dos tipos de procesos anteriores.

Proceso en Segundo Plano (*Background Process*)

Es un proceso que ejecuta una actividad no visible en ese momento para el usuario. Son procesos que pueden eliminarse cuando lo requiera el sistema al no tener influencia directa en las acciones que está realizando el usuario. Para ello, se emplea una lista conocida como LRU (*Least Recently Used*) de forma que primeros candidatos en ser eliminados son los procesos que hace más tiempo que no se mostraron en pantalla.

Proceso Vacío (*Empty Process*)

Se trata de un proceso que no recoge ningún bloque activo. Su fin es el de mejorar el tiempo de arranque cuando el componente tenga que ejecutarse en él a modo de caché. Por lo tanto, son los primeros procesos en ser eliminados para equilibrar los recursos en memoria.

2.1.1.5.3. Ciclo de vida de los principales componentes de una aplicación

El conocimiento del ciclo de vida de los componentes de una aplicación facilita su interacción con el usuario además de permitir una gestión correcta de los recursos del sistema y la creación de aplicaciones robustas.

Cada componente dispone de un ciclo de vida diferente, siendo los más importantes los de las actividades y servicios, los cuales se explican en este apartado.

Ciclo de vida de una actividad

El ciclo de vida de una actividad en Android [3] es diferente al de las aplicaciones en otros sistemas operativos debido a que en Android el ciclo de vida está controlado por el sistema y no por el usuario. Así, ciclo de vida de una actividad depende de su relación con las demás actividades y de la pila de actividades.

Una actividad en Android puede encontrarse en cuatro estados diferentes, los cuales se establecen en función de su posición en la pila de actividades:

- Activa (*Running*), cuando la actividad está la primera en la pila por lo que está visible y tiene el foco.
- Visible (*Paused*), en el caso de que la actividad esté visible pero no tenga el foco, es decir, cuando se pasa a otra actividad que no ocupa toda la pantalla o que deja ver la actividad anterior.
- Parada (*Stopped*), cuando la actividad no está visible en pantalla al estar oculta por otras actividades.
- Destruida (*Destroyed*), cuando la actividad termina al ejecutar el método *finish()* o porque es finalizada por el sistema, saliendo de la pila de actividades.

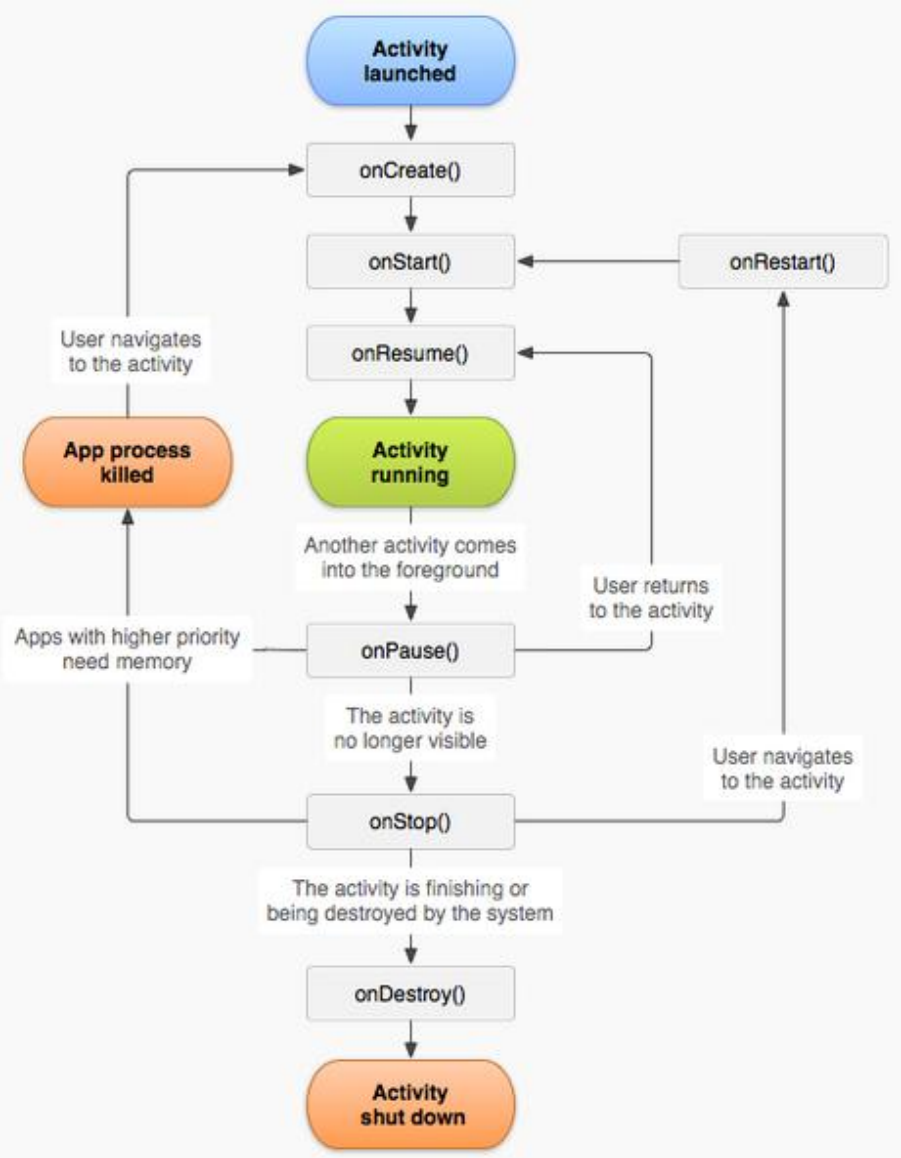

Figura 4. Ciclo de vida de una actividad [3]

Los cambios de estado en una actividad definen su ciclo de vida, representado en la Figura 4. Estos cambios producen eventos que pueden ser capturados por algunos

métodos de la actividad como *onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart(), y onDestroy()* los cuales ayudan a gestionar los cambios entre actividades. La descripción de la función de estos métodos se muestra abajo en la Tabla 1.

Método	**Utilización**
onCreate()	Invocado para la creación de la actividad
onStart()	Se llama para hacer visible la actividad
onResume()	Actúa antes de que la actividad comience a interactuar con el usuario
onPause()	Llamado cuando es otra la actividad que pasa a estado activo
onStop()	Se invoca cuando la actividad deja de ser visible
onRestart()	Invocado después de que la actividad se haya parado justo antes de volver a comenzarla.
onDestroy()	Llamado para la destrucción de la actividad

Tabla 1. Métodos del ciclo de vida de una actividad.

Ciclo de vida de un servicio

El ciclo de vida de un servicio [3] es más simple que el de una actividad. Básicamente, los servicios comienzan o se paran, aunque se deben estas dos acciones deben ser controladas debido a que los servicios pueden estar ejecutándose en segundo plano, sin intervención del usuario.

Todos los servicios deben extender de la clase *Service*. Los métodos más importantes que han de implementarse en ella son los explicados brevemente a continuación.

El ciclo de vida de un servicio puede tomar dos caminos diferentes según haya sido iniciado con *startService()* o con *bindService()*, ambos representados en la Figura 5.

En los servicios iniciados con *startService()*, el servicio es creado a partir de la llamada a dicho método desde otro componente. Una vez comenzado, el servicio puede ejecutarse en segundo plano de forma indefinida aunque el componente que lo lanzó haya sido destruido y lo hará hasta que el propio servicio llame a la función *stopSelf()* u otro componente llame a *stopService()* y el sistema lo elimine. Generalmente se tratan de servicios que realizan una sola operación y no devuelven resultados al componente que los llamó.

En el caso de los servicios iniciados mediante *bindService()*, el cliente es el que llama a la función y se comunica con ella a través de interfaz IBinder. De esta manera se establece una comunicación entre un cliente y un servicio, la cual puede darse por

terminada al utilizar el cliente la función *unbindService()*. Varios clientes pueden conectarse a un mismo servicio y el sistema no elimina el servicio hasta que hayan finalizado todas las comunicaciones establecidas.

Ambos caminos no son excluyentes, por lo que un cliente puede establecer comunicación con un servicio aunque este haya sido lanzado mediante *startService()*. No obstante, el servicio no finalizará hasta que no lo haga el cliente aunque se llamen a los métodos *stopService()* o *stopSelf()*.

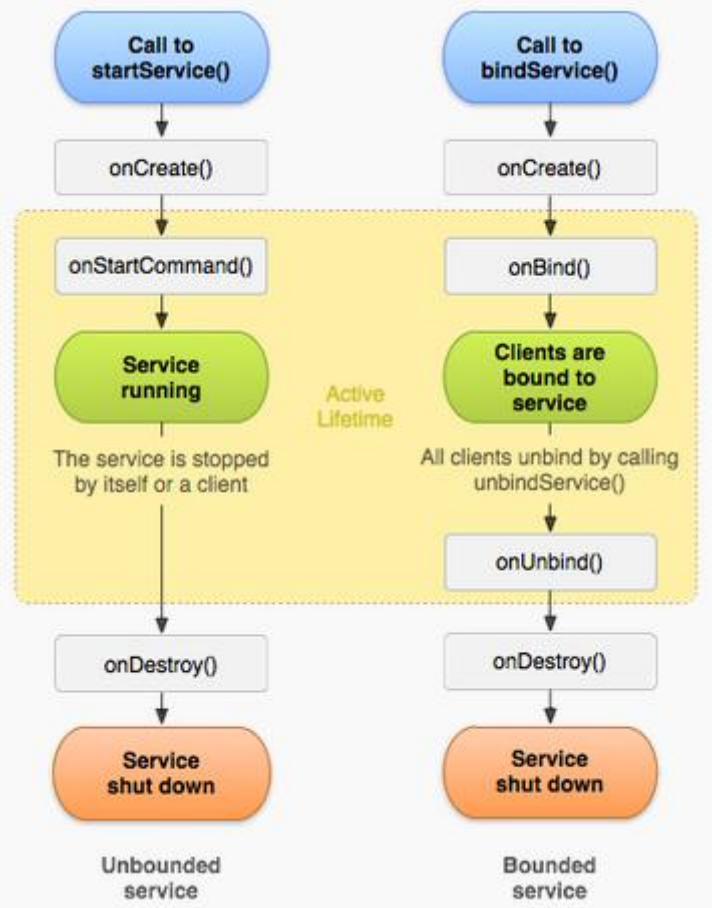

Figura 5. Ciclo de vida de un servicio [3]

2.1.1.5.4. Principales ficheros de una aplicación

Las aplicaciones basan su implementación en la utilización de varios ficheros. A través de ellos se recoge la estructura de la aplicación ya que definen sus componentes, recursos e interfaz de usuario. Los principales son el fichero *AndroidManifest.xml*, el archivo *main.xml* y *R.java*, cuyo cometido se explica seguidamente.

AndroidManifest.xml

El archivo *AndroidManifest.xml* es obligatorio en toda aplicación Android. En él se define la estructura y componentes de la misma, de forma que para que la aplicación pueda lanzar un nuevo componente es necesario que éste se encuentre declarado en el fichero. La declaración de los componentes viene acompañada de más información sobre la aplicación, como los permisos de los que dispone para interactuar con otras así como los permisos necesarios para que otros puedan hacerlo lo propio con ella, o declarar el nivel mínimo de API de Android que requiere.

main.xml

El fichero conocido como *main.xml* o *layout* de una actividad contiene el diseño de los diferentes elementos mostrados por pantalla al usuario de la actividad. Generalmente, existen varios ficheros de este tipo, uno por actividad, los cuales se almacenan en la carpeta */res/layout* del proyecto[2]. Este archivo provee de una interfaz gráfica para facilitar su creación, a cuyos componentes puede darse después utilidad mediante código programado en Java.

R.java

El fichero *R.java* es generado automáticamente por el sistema y que no debe editarse. Este archivo contiene los punteros a los recursos empleados, que son todos aquellos que no forman parte del código Java, tales como cadenas de texto, archivos, *layouts* o imágenes y los cuales se encuentran almacenados en diferentes subcarpetas dentro de la carpeta */res* del proyecto.

2.1.1.6. Seguridad

Android dispone de un sistema de seguridad no centralizado en concordancia con su filosofía de *software* libre, es decir, no existe un mecanismo central encargado de controlar y verificar la seguridad. La seguridad en Android [5] se basa en tres medidas, la ejecución de las aplicaciones en procesos independientes, la concesión de permisos para la utilización de recursos y la firma de las mismas.

[2] La carpeta */res* contiene la mayoría de los recursos (ficheros multimedia, diseños de pantalla, cadenas, etc.) que se utilizan en un proyecto Android.

Ejecución en procesos independientes

En primer lugar, Android aprovecha la seguridad que le proporciona su núcleo Linux, gracias al cual puede controlar el acceso de las aplicaciones al *hardware* y a los recursos de otras aplicaciones. De esta manera, las aplicaciones se ejecutan dentro de procesos independientes que no pueden interactuar con otras excepto cuando se haya declarado explícitamente permiso para ello (Figura 6).

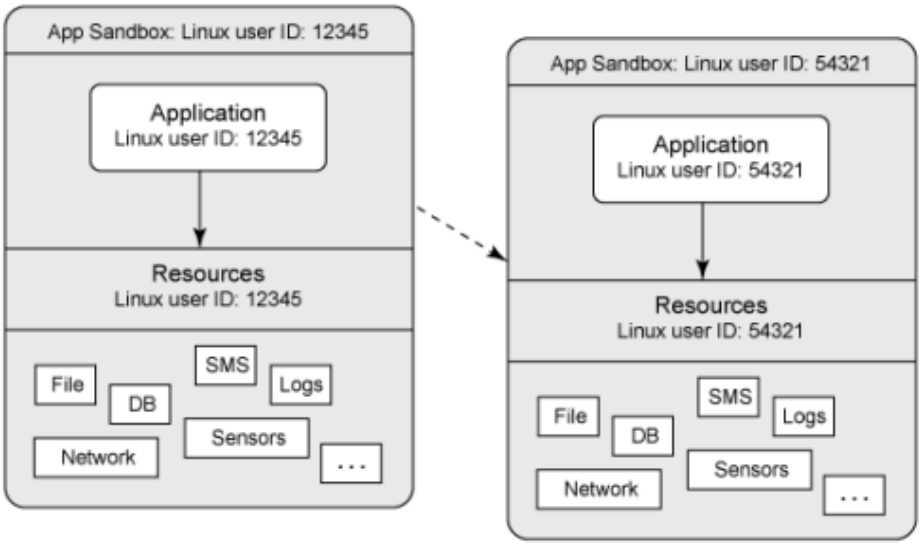

Figura 6. Ejemplo de dos aplicaciones de Android con sus propios procesos o recintos de seguridad [5]

Android mantiene una cuenta de usuario Linux (*user* ID) por cada aplicación instalada en el sistema. Los datos almacenados por la aplicación son asociados a su usuario Linux de forma que no puedan ser accedidos por otras aplicaciones. Sin embargo, sí que existe la posibilidad de otorgar modos de acceso en la creación de ficheros, permitiendo su lectura o escritura por parte de diferentes aplicaciones, aunque el fichero tendrá un único propietario que será el usuario Linux de la aplicación que lo creó.

Concesión de permisos

La segunda de las medidas de seguridad se pone de manifiesto en el caso de que una aplicación necesite acceder a datos o componentes del sistema que puedan poner en un compromiso a la seguridad del mismo. Para ello, se deben conceder permisos. Android dispone de un sistema de permisos para el control del acceso a los recursos y características especiales del *hardware*. Estos permisos son declarados por el desarrollador de la aplicación en el fichero *AndroidManifest.xml*.

Las aplicaciones tienen la obligación de informar sobre los recursos que tienen intención de usar a través de sus paquetes de instalación. Por tanto, el control del comportamiento de las aplicaciones y servicios en Android solo puede ser llevado a cabo por el usuario en el momento previo a su instalación, a través de la aceptación de las políticas de cada aplicación.

Mediante este sistema de concesiones se controla que una aplicación no haga uso de permisos no manifestados, interrumpiéndose y generándose una excepción de permiso en caso de que pretenda acceder a un recurso no declarado en su instalación. Sin embargo, este mecanismo proporciona un control reducido sobre los permisos ya que no es posible conocer cómo se están utilizando por parte de la aplicación tras su aceptación. Además, la decisión sobre la concesión de los permisos es estática, es decir, no existe la posibilidad de que los permisos otorgados puedan ser modificados posteriormente una vez que la aplicación ha quedado instalada. También se debe señalar que Android no permite establecer políticas por las que se conceda a una aplicación el acceso a un recurso un número fijo de veces o solamente bajo ciertas circunstancias [6].

Firma de una aplicación

Por último y como medida disuasoria de creación de *software* maligno, toda aplicación debe ser firmada con un certificado digital que identifique a su creador. Igualmente, tras la modificación de una aplicación, ésta debe ser firmada de nuevo de forma que sólo el propietario de la clave privada podrá realizar modificaciones en ella.

La firma de las aplicaciones en Android es simple y puede ser realizada de forma autónoma, es decir, no es condición necesaria la firma del certificado digital por parte de una autoridad de certificación, lo que facilita y reduce costes en el proceso de publicación de las mismas [6].

2.1.1.6.1. Seguridad basada en contextos

Una forma de proporcionar un control más específico sobre la configuración de seguridad en Android es incorporar el uso de contextos. La seguridad por contexto [7] se trata, por tanto, de una política de seguridad cuyos requisitos tienen en cuenta el contexto en el que se encuentra el dispositivo.

Los contextos de seguridad pueden definirse considerando diferentes variables de entorno, como la localización, el tiempo, la temperatura, la luz o el ruido, la presencia

de otros dispositivos, de puntos de acceso Wifi o de una combinación de varios de ellos. Éstos son de fácil obtención a partir de los numerosos sensores con los que cuentan los terminales móviles actuales. La monitorización de estos factores permite establecer el nivel de familiaridad y dividir es estado del sistema en una serie de casos de seguridad para aplicar las medidas de seguridad correspondientes. Estas medidas pueden estar relacionadas con el control de acceso del dispositivo o el cambio de bloqueo en el mismo o de aplicaciones entre otras.

Algunas de las posibles aplicaciones del uso de contextos de seguridad [7] y su relación con las variables de entorno son, el control de acceso a cierta información en el dispositivo en el caso de prestar el dispositivo a un familiar, el cual estaría relacionado con el reconocimiento del usuario mediante la cámara delantera del terminal o el bloqueo/desbloqueo del terminal en función de si el usuario se encuentra, por ejemplo, en un aeropuerto o en su hogar, lo que está relacionado con su información de localización.

La detección y el uso de contextos debe realizase de forma transparente ya que, una vez definidos o creados a partir de una sucesión de encuentros repetidos, éstos son detectados automáticamente y su configuración de seguridad es aplicada directamente sin intervención del usuario. Esto elimina la necesidad de utilizar políticas por defecto que pueden no corresponderse con el caso en el que se encuentra cada dispositivo. Además, el empleo de contextos simplifica el control sobre la seguridad del terminal al abstraer al usuario de la configuración poco intuitiva de los parámetros de seguridad del mismo.

2.1.1.7. Almacenamiento de datos

En la mayoría de casos, las aplicaciones necesitan guardar sus datos de forma permanente. Para ello se pueden utilizar diferentes técnicas de almacenamiento [5] las que, además de conservar la información relevante, permiten que sea posible la compartición de estos datos con otras aplicaciones o usuarios. A continuación se presentan las técnicas más empleadas.

- **Preferencias**: Es un mecanismo ligero, proporcionado a través de la clase *SharedPreferences*, que posibilita el almacenamiento y recuperación de datos primitivos en parejas de clave y valor. Es por esto que generalmente se utiliza para guardar los parámetros de configuración de las aplicaciones.

- **Ficheros**: Consiste en el almacenamiento de ficheros tanto en la memoria interna del dispositivo como en un medio removible, cuyo dispositivo más empleado es la tarjeta SD. Su acceso es similar a los ficheros en Java mediante la utilización de *inputs* y *outputs streams.*
- **XML**: Se refiere a los ficheros que siguen este estándar. XML es uno de los estándares más empleados actualmente para la codificación de información sin embargo, leer y escribir ficheros XML es muy laborioso. Para facilitar la manipulación de este tipo de archivos, Android dispone de librerías SAX (Simple API for XML) y DOM (Document Object Model).
- **Base de datos**: Android permite la creación y utilización de base de datos SQLite a través de sus APIs de una forma sencilla y empleando pocos recursos del sistema. SQLite es uno de los lenguajes de programación más utilizados para bases de datos debido a su portabilidad, rendimiento y estabilidad. Además cabe destacar la longitud variable de sus registros como otra de sus ventajas. Disponer de registros de datos ajustados a su tamaño real permite obtener bases de datos más reducidas que optimizan la velocidad de búsqueda en ellas.

  Para manipular bases de datos en Android se debe extender de la clase *SQLiteOpenHelper* cuyos métodos nos facilitan la creación, actualización y borrado de las mismas. Esta clase tiene se ocupa de abrir la base de datos si existe o de crearla en caso contrario así como de actualizar la versión de la misma si se modifica su estructura. Además, cuenta con los métodos necesarios para abrir la base de datos en modo lectura o lectura y escritura.

  Cada base de datos es privada para una aplicación pero accesible desde todas las clases de ésta y para su compartición entre aplicaciones es necesario el uso de proveedores de contenidos.

  Por todas estas ventajas, el uso de base de datos en Android se convierte en un potente pero sencillo mecanismo de almacenamiento.
- **Internet**: Es la opción referida al empleo de una conexión de Internet para el almacenamiento y recuperación de datos desde la nube.

2.1.2. Servlet

2.1.2.1. Introducción

Un Servlet es un objeto del lenguaje de programación Java que permite la generación de contenido Web dinámico. Aunque los Servlets pueden responder a cualquier tipo de solicitud, su principal uso es extender las capacidades de un servidor Web. Para ello, se ejecutan en un contenedor Web en el servidor sin necesidad de ninguna clase de interfaz gráfica.

El protocolo HTTP, sobre el cual se construyen las transferencias de información a través de la Web, basa su funcionamiento en un modelo de solicitud-respuesta. Así, un Servlet ejecutándose en un servidor Web recibe una petición HTTP procedente de un navegador u otro cliente HTTP, la procesa y devuelve la correspondiente respuesta. Estos procesos de petición y respuesta están automatizados gracias a la API Servlet que define clases específicas HTTP.

2.1.2.2. Historia

Inicialmente la interacción en la Web era notablemente reducida. Ésta se realizaba a través de código HTML (HyperText Markup Language) y elementos embebidos de JavaScript y la utilización de Java estaba asociada a pequeños programas transferidos con las páginas Web y ejecutados en el cliente, conocidos con el nombre de Applets. Éstos eran una forma potente de introducir programas más elaborados dentro de una página Web.

Sin embargo, pronto surgió la necesidad de procesar datos también en los servidores, para lo que se comenzó a emplear el estándar CGI (Common Gateway Interface). Este estándar se encargaba de comunicar un programa llamado CGI con un servidor Web a través de un formulario de acceso en el navegador. No obstante, se trataba de una solución con un alto consumo de recursos ya que necesitaba de la creación de un nuevo proceso con cada petición. Además, para un número de peticiones simultáneas al mismo programa, su código debía ser cargado en memoria el mismo número de veces.

La primera especificación de un Servlet, conocida como *Jeeves*, fue desarrollada por Sun Microsystems en 1997. Gracias a la JVM (Java virtual Machine), los Servlets crean un nuevo hilo o subproceso ligero de Java con cada petición evitando, a diferencia de los CGIs [8], la sobrecarga de procesos pesados en el sistema. Esta característica ha

contribuido a que los Servlets se hayan establecido como una solución óptima para la interacción en la Web y a que hoy en día sean una herramienta ampliamente utilizada en el desarrollo de aplicaciones para servidores Web además de por su por su trabajo transparente frente a los formulario HTML y su completa interacción con los Applets.

2.1.2.3. Características

Los Servlets cuentan con varias características que los hacen idóneos para su utilización en servidores Web. A continuación, se citan algunas de las principales [9].

La primera de sus características, como ya se ha adelantado, es su rapidez y eficiencia. Los Servlets tienen una alta velocidad de respuesta debido al uso de hilos para atender a las peticiones. Asimismo, consumen menos recursos ya que cada Servlet es cargado en memoria una sola vez en lugar de una vez por solicitud.

Los Servlets pueden interactuar con otros Servlets, tanto si están situados en la misma máquina como en otra remota. Esto permite que se pueda distribuir la carga de trabajo de una misma aplicación Web entre varios Servlets mejorando los tiempos de interacción con el cliente.

Al estar basados en Java, otra de sus características es su portabilidad entre plataformas. Los Servlets son independientes del servidor utilizado y de su sistema operativo por lo que el servidor puede estar escrito en cualquier otro lenguaje de programación. Asimismo, disponen de una API Servlet que hace más potente y sencilla su implementación.

Otra de sus características es que los Servlets pueden obtener información permitida por el protocolo HTTP (Hypertext Transfer Protocol) acerca del cliente. Ésto junto al uso de *cookies*, permite el almacenamiento de dicha información para una mejor interacción entre el servidor y el cliente, cuyo ejemplo más claro es el mantenimiento de sesiones.

Por último cabe destacar la seguridad que proporcionan. Al tratarse de programas ya compilados corren menor riesgo de introducir ataques de ejecución de comandos no deseados en el servidor. Además, un Servlet puede proporcionar seguridad a otros servidores actuando como *proxy* de un Applet, controlando el acceso de los mismos a otros servidores de datos.

2.1.2.4. Arquitectura y funcionamiento

La API Servlet es un conjunto de clases e interfaces para la implementación de la interfaz entre el servidor y los Servlets definidos en los paquetes *javax.servlet*, que contiene clases genéricas, válidas para cualquier protocolo y en el paquete *javax.servlet.http* concretamente para el protocolo HTTP. Todos los Servlets para servidores Web deben implementar la interfaz Servlet y los Servlets para servidores Web lo hacen mediante la extensión de la clase *HttpServlet*, una subclase de la clase *GenericServlet*, que implementa la interfaz Servlet.

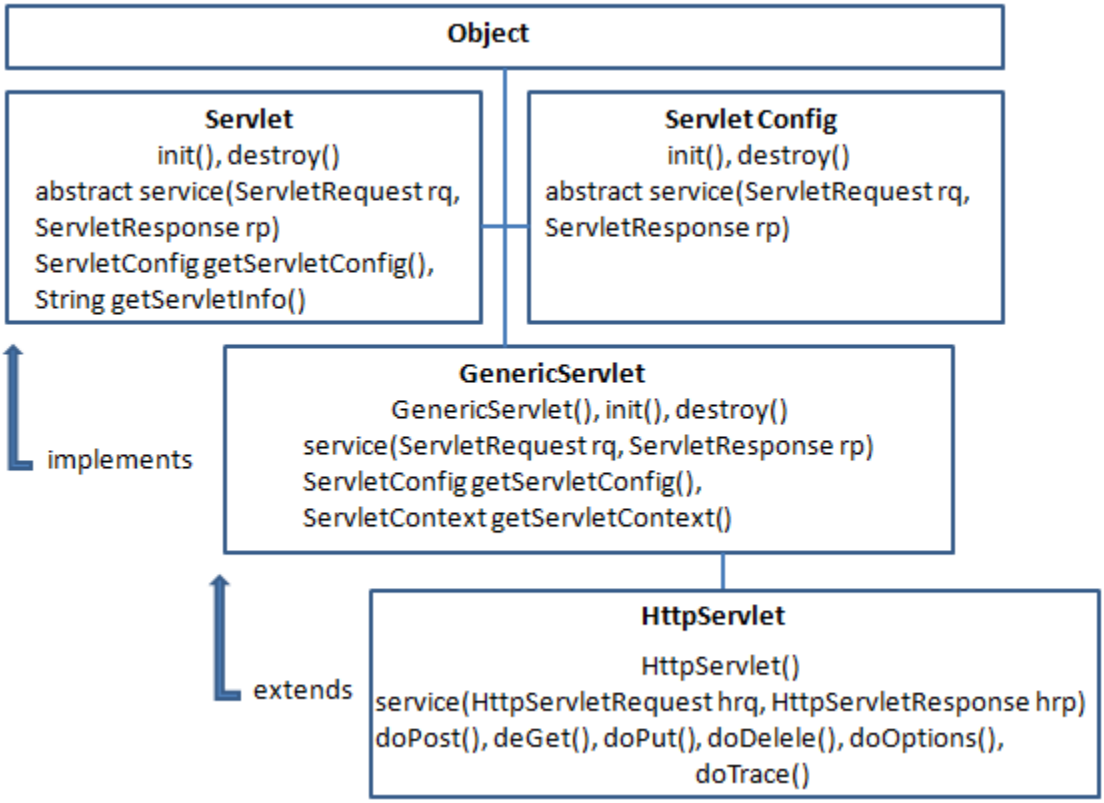

Figura 7. Jerarquía, herencia y métodos de las clases para la creación de Servlets [9]

La interfaz Servlet define los métodos para la comunicación entre un Servlet y un cliente. Los principales métodos son *init()*, *service()* y *destroy()*, los cuales definen el ciclo de vida del Servlet. Para la interacción desde el cliente hasta el servidor se utiliza el objeto *ServletRequest*, mientras que *ServletResponse* es el encargado de encapsular la comunicación en sentido inverso, y que en el caso concreto del protocolo HTTP se corresponden con *HttpServletRequest* y *HttpServletResponse* respectivamente.

El funcionamiento del mecanismo de intercambio mediante el uso de Servlets (Figura 8) es el explicado a continuación. En primer lugar, el cliente realiza una petición HTTP al

servidor (1) a través de un navegador o cualquier otra aplicación con acceso a la Web. Al tratarse de una petición HTTP, ésta invoca al servidor a través del método GET o POST. La petición es de la forma *http://hostname:port/context/nombre_servlet*. Además, se pueden enviar parámetros con sus valores añadiéndolos al final y resultando *http://hostname:port/context/nombre_servlet?param1=valor1&param2=valor2*.

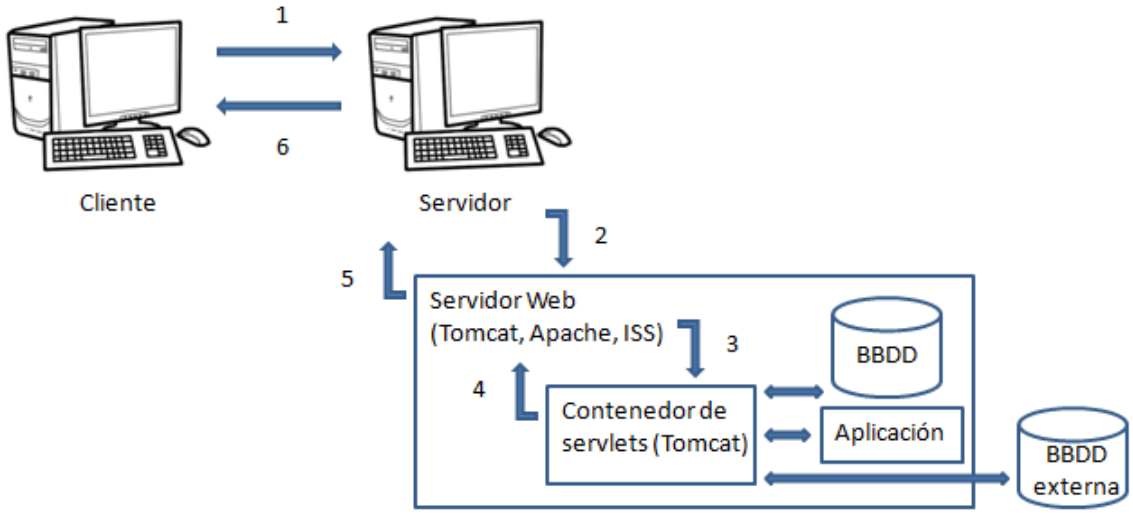

Figura 8. Funcionamiento de un Servlet

La petición llega al servidor y el servidor Web la recoge junto con sus parámetros (2). Tras interpretar la petición, el servidor Web detecta que hay que ejecutar un Servlet por lo que delega dicha acción al contenedor de Servlets (3), aunque puede darse el caso de que el contenedor de Servlets también realice las funciones de servidor Web. Éste se encarga del proceso de obtención de la información solicitada tanto si esta se encuentra dentro del servidor como si se encuentra fuera de él, mediante consultas a bases de datos u otras aplicaciones.

Cuando el Servlet termina su ejecución, devuelve el resultado de la misma al servidor Web (4) y el servidor Web envía este resultado a través del servidor, para lo que se emplea un objeto de la clase *PrintWriter*. Finalmente, la respuesta HTTP llega al cliente (6).

2.1.2.5. Ciclo de vida de un Servlet

Desde la inicialización de un Servlet hasta su destrucción, éste pasa por varios estados según las situaciones en las que se encuentre. Estos estados forman su ciclo de vida, cuyo esquema es el mostrado en la Figura 9.

Cuando el servidor recibe la primera petición a un Servlet se ejecuta el método *init()*, a través del que se llevan a cabo las tareas de inicialización como, por ejemplo, la inicialización de las variables globales del Servlet, de archivos o parámetros de configuración como conexiones con bases de datos. El método *init()* sólo volverá a ser llamado o en caso de que se vuelva a recargar el Servlet, lo cual no puede ocurrir sin antes terminar con el Servlet en ejecución.

Una vez el Servlet está preparado, se ejecuta su método *service()*. Cada petición recibida se ejecuta en un hilo nuevo, aunque es posible indicar que se atiendan todas desde el mismo. En el caso de tratarse de un Servlet HTTP, los métodos ejecutados serán *doGet()* y *doPost()* dependiendo de si las peticiones se realizaron con GET o POST respectivamente.

Los Servlets están en continua ejecución hasta que dejan de ser necesarios y se procede a su terminación. Finalmente, cuando un servidor destruye un Servlet se ejecuta el método *destroy()* de forma que se eliminan las variables creadas durante el estado de inicialización y cerrar las conexiones que se hayan abierto.

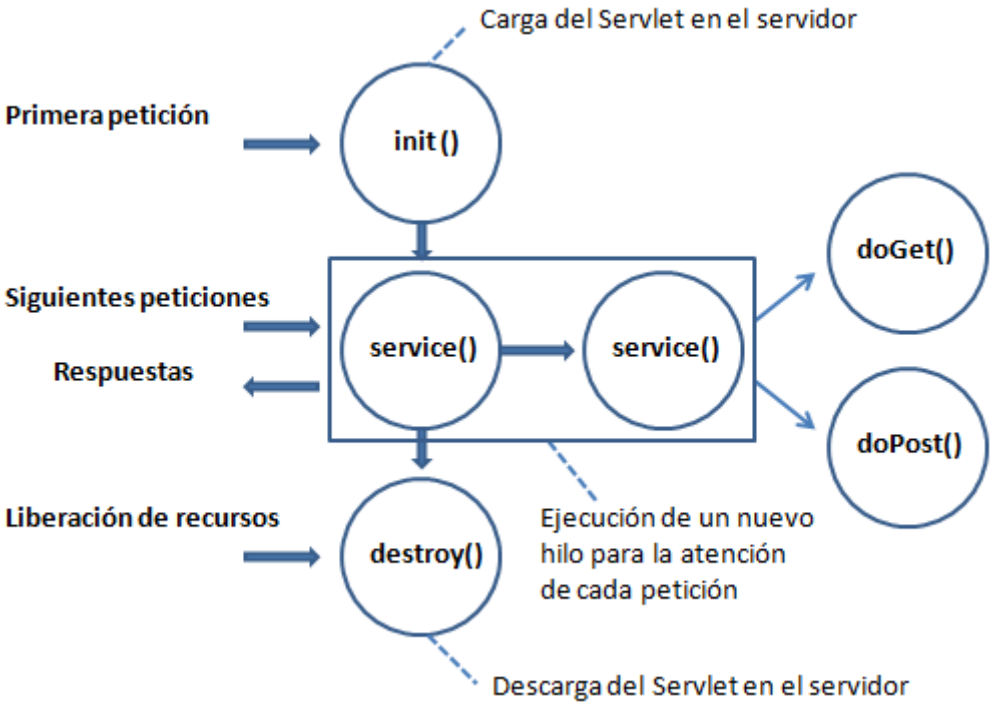

Figura 9. Ciclo de vida de un Servlet

2.1.2.6. Contenedor de Servlets

Un contenedor de Servlets [10] es un programa que se encarga de recibir las peticiones que llegan a un servidor y de redireccionarlas a un objeto Servlet para su procesamiento y generación de la respuesta.

La utilización de los contenedores de Servlets, junto al motor Servlet, oculta al programador Web los detalles de la conexión a la red, así como la obtención de los parámetros de las solicitudes. El contenedor más utilizado es Tomcat, seguidamente explicado.

2.1.2.6.1. Tomcat

Tomcat [11] es un contenedor de Servlets desarrollado por la Apache Software Foundation bajo el proyecto Jakarta y la licencia Apache 2.0 que implementa la tecnología Servlet y las JSP (JavaServer Pages) de Sun Microsystems. El desarrollo de Tomcat es llevado a cabo por miembros de la fundación y voluntarios independientes que disponen de acceso al código fuente bajo los términos que la Apache Software Foundation establece.

Tomcat se trata de un servidor Web sencillo al que se accede a través del protocolo HTTP y que a menudo se presenta en combinación con el servidor Web Apache, cuya unión se conoce como Apache Tomcat. Además, incluye el compilador Jasper que compila las JSP para convertirlas en Servlets.

Apache Tomcat es utilizado como servidor Web autónomo para un elevado nivel de tráfico y de alta disponibilidad. Además, al estar desarrollado en Java, Tomcat tiene la ventaja de funcionar en cualquier plataforma que disponga de una JVM.

Las primeras versiones distribuidas de Tomcat fueron las 3.0.x y las más recientes son las 7.x. A partir de la versión 4.x, Tomcat incluye sus tres componentes principales. En primer lugar, el contenedor de Servlets “Catalina”, que es el encargado de la implementación del Servlet y JSPs, el contenedor HTTP “Coyote”, como conector para la conexión a través del protocolo HTTP y para el envío de la solicitud al motor Tomcat para que éste la procese, y por último, un motor para JSP conocido como “Jasper” y que es el encargado del análisis de los archivos JSP para compilar el código Java. Las principales características de las versiones de Tomcat desarrolladas con las de la Tabla 2, mostrada a continuación.

Versión	Características
Tomcat 3.x	Implementado a partir de las especificaciones Servlet 2.2 y JSP 1.1 Recarga de Servlets Funciones básicas HTTP
Tomcat 4.x	Implementado a partir de las especificaciones Servlet 2.3 y JSP 1.2 Contenedor de Servlets Catalina Motor JSP Jasper Conector Coyote Java Managment Extensions (JMX) y administración basada en Struts
Tomcat 5.x	Implementado a partir de las especificaciones Servlet 2.4 y JSP 2.0 Recolección de basura reducida Capa envolvente nativa para Windows y Linux para la integración de las plataformas Análisis rápido JSP
Tomcat 6.x	Implementado a partir de las especificaciones Servlet 2.5 y JSP 2.1 Soporte para Unified Expression Language 2.1 Diseñado para funcionar en Java SE 5.0 y posteriores Soporte para Comet s través de la interfaz CometProcessor
Tomcat 7.x	Implementado a partir de las especificaciones Servlet 3.0 y JSP 2.2 Mejoras para la detección y prevención de fugas de memoria en las aplicaciones Web Limpieza interna de código Soporte para la inclusión de contenidos externos directamente en una aplicación Web

Tabla 2. Evolución y características principales de las diferentes versiones de Tomcat [11]

2.2. Tecnologías relacionadas con la gestión del riesgo

2.2.1. NVD (National Vulnerability Database)

2.2.1.1. Introducción

La *National Vulnerability Database* (NVD) es el repositorio público de estándares para la gestión de datos de vulnerabilidades del gobierno de los Estados Unidos. Estos datos permiten la automatización de la gestión de las vulnerabilidades y obtener medidas de seguridad y conformidad. La NVD está proporcionada por el NIST, agencia de la Administración de Tecnología del departamento de Comercio estadounidense.

La NVD proporciona diversas bases de datos con *checklists*, vulnerabilidades de seguridad del software, errores de configuración, identificadores de productos tecnológicos y métricas de impacto.

Al tratarse de un repositorio público, se pueden realizar consultas sobre las vulnerabilidades a través de su buscador de vulnerabilidades. Asimismo, es posible descargar la información completa de las vulnerabilidades en ficheros XML. Estos ficheros se encuentran disponibles en [12].

2.2.1.2. Principales recursos disponibles

La NVD se basa en varios recursos estandarizados para hacer adecuada su utilización e interpretación en diferentes sistemas. Entre ellos destacan el esquema de nombrado CPE (*Official Common Platform Enumeration*), el diccionario CVE (*Common Vulnerabilities and Exposures*), los códigos CWE (*Common Weakness Enumeration*) y las métricas CVSS (*Common Vulnerability Scoring System*). A continuación se explica cada uno de estos recursos y su aportación a la NVD.

2.2.1.2.1. CPE

CPE es un esquema estructurado de nombres para sistemas tecnológicos basado en la sintaxis genérica de URIs (Uniform Resource Identifier). Así, se trata de una forma estandarizada de describir e identificar tipos de aplicaciones, sistemas operativos y *hardware*. Los nombres CPE o CPE-IDs recogen un formato de nombre formal, un método de comprobación de nombres en un sistema y un formato de descripción de texto vinculante [12].

El diccionario CPE es la colección oficial de los nombres CPE. Se trata de un recurso proporcionado en formato XML y que está publicado y mantenido por el NIST, el cual emplea para su NVD. El NIST se encarga a su vez de comprobar si éste cumple con las especificaciones CPE sobre el nombrado, macheado, diccionario y aplicabilidad del lenguaje, y de la gestión de su contenido y garantía de calidad. Asimismo, el NIST utiliza los nombres CPE en su programa SCAP (*Security Content Automation Protocol*) [12], el cual combina un conjunto de herramientas para la automatización de la gestión de vulnerabilidades y evaluación del cumplimiento de los requisitos de seguridad tecnológica.

El diccionario, además de servir como repositorio, tiene como cometido enlazar metadatos descriptivos a los nombres CPE, tales como títulos y notas, y también enlazarlos pruebas de diagnóstico, como comprobaciones para determinar si una determinada plataforma coincide con el nombre especificado [13].

2.2.1.2.2. CVE

El CVE es un diccionario gratuito, público y de alcance internacional para la identificación de vulnerabilidades y exposiciones relacionadas con la seguridad, impulsado por el MITRE[3]. Varias son las organizaciones que proveen de información sobre vulnerabilidades al CVE. Esta información llega a través de observaciones a partir de las bases de datos de vulnerabilidades del origen de esos datos o bien a través de listas en herramientas de evaluación o resúmenes periódicos de vulnerabilidades. El CVE está formado por identificadores CVE, comúnmente designados como CVE-IDs, que son unos identificadores únicos y comunes para el conocimiento público de vulnerabilidades de seguridad. Cada CVE-ID incluye su número de identificación, una breve descripción de la vulnerabilidad o exposición de seguridad y cualquier referencia pertinente relacionada con ella. En la Figura 10 se muestra el ejemplo de un CVE-ID. Estos identificadores facilitan el intercambio de datos haciéndolo compatible entre diferentes productos de seguridad. Además, proporcionan una guía para la evaluación de la cobertura de estos servicios y herramientas.

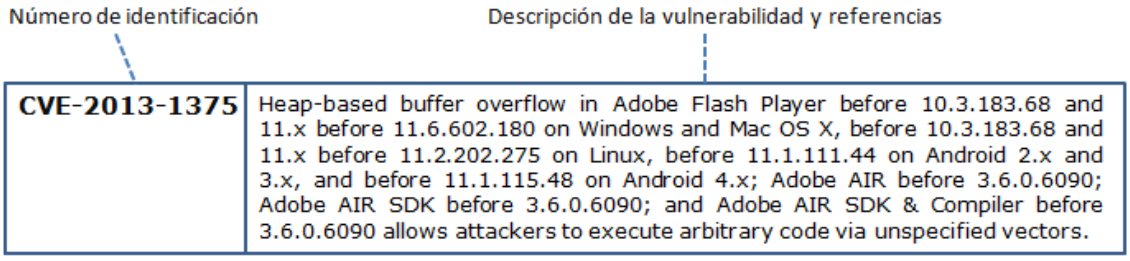

Figura 10. Ejemplo de un CVE-ID

La NVD utiliza los identificadores proporcionados por el CVE para su motor de búsqueda de vulnerabilidades así como fuente de información. De hecho, la funcionalidad completa de la base de datos para la lista de CVE se proporciona

[3] El MITRE es una organización sin ánimo de lucro fundada en 1958 que trabaja en asuntos de interés público. Para ello utiliza su experiencia en ingeniería de sistemas, información tecnológica, conceptos operacionales y modernización empresarial con el fin de tratar las necesidades de sus sponsors.

mediante la asociación de MITRE con la NVD a través de la cual se realizan la búsquedas [14].

2.2.1.2.3. CWE

El CWE provee de un lenguaje para la agrupación de un conjunto unificado de debilidades *software* de seguridad con fin de mejorar el tratamiento de las causas de las vulnerabilidades y conseguir un análisis más eficaz de las mismas. Está proporcionado por el MITRE con el apoyo del DHS[4] y es de uso público [15].

Todos los códigos CWE forman parte de una estructura jerárquica a partir de la que se obtienen varios niveles de abstracción. Una parte de esta jerarquía es la representada en la Figura 11. Los CWE que se encuentran en los niveles superiores proporcionan una visión general de un tipo de vulnerabilidad, ramificándose en tipos más concretos dentro de la misma. En contraposición, los niveles más bajos en la estructura proporcionan una mayor granularidad, teniendo menor número de descendientes o ninguno.

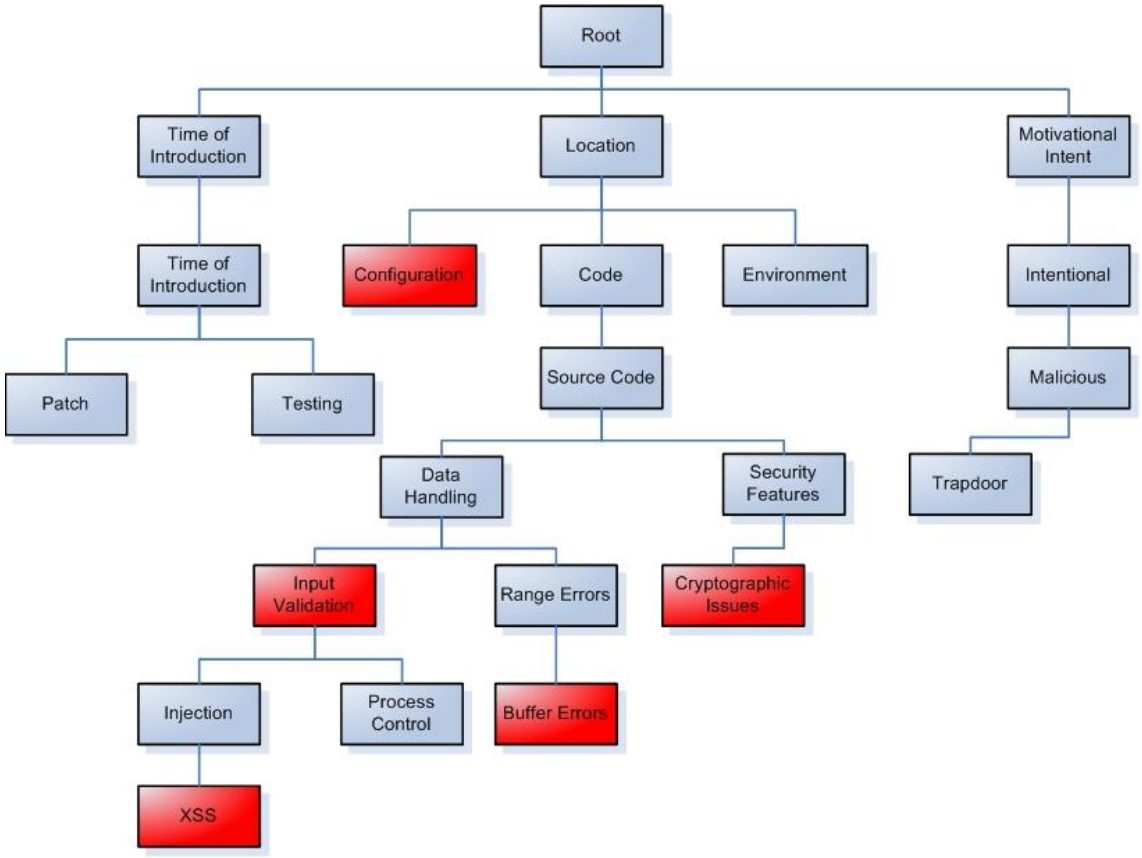

Figura 11. Porción mapeada en la NVD de la estructura CWE [16]

[4] El DHS (Department of Homeland Security) es el departamento del gobierno estadounidense para la protección de Estados Unidos frente a ataques terroristas u otros accidentes o catástrofes.

La NVD utiliza el CWE como mecanismo de clasificación para la diferenciación de los CVEs a partir del tipo de vulnerabilidad con la que están relacionados, para lo que incorpora el CWE en la puntuación de las mismas. No obstante, solamente utiliza una sección de la estructura general CWE, en el caso de la Figura 11, se trata de los cuadros coloreados de rojo.

En la página del MITRE puede consultarse una lista detallada con los códigos, en la que se explica cada CWE detalladamente. La Tabla 3, mostrada a continuación, contiene los códigos de los CWE utilizados en la NVD junto con su nombre y descripción.

Nombre	CWE-ID	Descripción
Asuntos de autenticación	CWE-287	Fallo en la autenticación de usuarios
Gestión de credenciales	CWE-255	Fallo en la creación, almacenamiento, transmisión o protección de contraseñas u otras palabras clave
Permisos, privilegios y control de acceso	CWE-264	Fallo en la aplicación de permisos u otras restricciones de acceso a recursos o problema en la gestión de privilegios
Errores de búfer	CWE-119	Desbordamiento de buffer y otros errores relacionados con los límites de los búferes en los que un programa o aplicación intenta añadir más datos que los que soporta su capacidad o cuando se intentan guardar datos fuera de los límites de memoria del búfer.
Falsificación de petición en sitios cruzados (CSRF)	CWE-352	Fallo al no verificarse que el remitente de una petición web en realidad tiene la intención de hacerlo. Los atacantes pueden enviar comandos no autorizados a través de victimas haciéndose pasar por ellas y engañando al sitio web que confía en ellas. CSRF se asocia generalmente con XSS, pero se trata de otro asunto.
Secuencia de comandos en sitios cruzados (XSS)	CWE-79	Fallo en la validación de un sitio de forma que es posible la inyección de código en páginas web vistas por un usuario
Asuntos de criptografía	CWE-310	Algoritmos inseguros o inapropiado uso de ellos; Implementación incorrecta de un algoritmo que compromete la seguridad; Falta de encriptación (texto plano); Claves o gestión de certificados débiles, divulgación de claves, problemas en la generación de números aleatorios
Ruta transversal	CWE-22	Los usuarios tienen la posibilidad de accede a cualquier directorio superior sin control.

Inyección de código	CWE-94	Provoca que un sistema lea y ejecute un fichero controlado por atacantes. Incluye inclusiones remotas en ficheros PHP, subida de ficheros con extensiones ejecutables, inserción de código en ficheros ejecutables y otros
Vulnerabilidad en el formato de cadenas	CWE-134	Control de información de entrada por parte de atacantes como el formato de cadenas de entrada en determinadas funciones
Configuración	CWE-16	Problema general de configuración que no está asociados con problemas de contraseñas o permisos
Fuga de información / Divulgación	CWE-200	Exposición de información del sistema, información sensible o privada, etc.
Validación de entrada	CWE-20	Fallo al asegurarse de que la entrada está formada adecuadamente y contiene datos válidos respecto a las especificaciones de un programa o aplicación. Nota: esto solapa con otras categorías como XSS, Errores numéricos e Inyección SQL
Errores numéricos	CWE-189	Desbordamiento de enteros, truncado, signos, y otros errores que puede ocurrir cuando se manejan con números
Inyección de comandos de SO	CWE-78	Permiso para la inyección de líneas de comando que invocan a otros programas, utilizando *system()* o funciones similares en la entrada del usuario
Condiciones de carrera	CWE-362	El estado de un recurso puede cambiar durante el tiempo que transcurre entre que se comprueba y se accede a él
Errores en la gestión de recursos	CWE-399	El software permite a atacantes consumir recursos de forma masiva como memoria, CPU, espacio en disco, etc.
Inyección SQL	CWE-89	Cuando la entrada del usuario puede estar incorporada en sentencias SQL sin ser filtradas adecuadamente, permitiendo la ejecución de comandos SQL
Seguimiento de enlaces	CWE-59	Fallo en la protección contra el uso de enlaces duros (*hard links*) que pueden apuntar a ficheros que no deberían ser accedidos desde el programa o aplicación
Otros	*No Mapping*	NVD solo utiliza un subconjunto del mapeo completo de CWE. El tipo de debilidad no está contemplada en ese subconjunto
No contemplados en CWE	*No Mapping*	Tipos de debilidades no cubiertas en la versión de CWE utilizada para el mapeo

Tabla 3. Conjunto de CWEs asignados a vulnerabilidades en la NVD [16]

2.2.1.2.4. CVSS

El CVSS se trata de un sistema de medición estándar y preciso de vulnerabilidades que permite obtener una puntuación general para cada vulnerabilidad en función de una

serie de métricas. De esta forma, se puede utilizar para establecer prioridades a la hora de su resolución y permite conocer las características a partir de las cuales se generaron los resultados relacionados con una vulnerabilidad. La NVD provee de las puntuaciones CVSS de todas las vulnerabilidades que recoge, utilizando para ello la puntuación CVSS versión 2 [17].

Este sistema se compone de tres grupos de métricas, el grupo de métricas de base, temporales y de entorno. El grupo de métricas de base tiene como fin definir las características fundamentales de una vulnerabilidad proporcionando una representación general de la misma, mientras que grupo de métricas temporales y de entorno, proveen de información contextual que muestra el riesgo en su entorno con una mayor precisión.

Cada uno de los grupos de métricas contiene una serie de indicadores, los cuales pueden observarse en la Figura 12. Sin embargo, la NVD sólo emplea el primero de los grupos en la descripción de las vulnerabilidades que almacena debido a que el resto de métricas dependen de factores externos a la vulnerabilidad, aunque sí que dispone de una herramienta de cálculo de puntuación CVSS que incorpora datos temporales y del entorno [18].

Figura 12. Grupos de métricas del CVSS [17]

Las puntuaciones globales o *scores* de cada vulnerabilidad se calculan a partir de sus métricas de base. Una vez se asigna los valores a sus indicadores, la ecuación de base obtiene una puntuación comprendida entre 0 y 10 y se crea un vector que contiene los valores asignados a cada métrica de forma que puede conocerse los valores a partir de los que se ha obtenido la puntuación general. Por este motivo el vector debe acompañar siempre a la puntuación de la vulnerabilidad, y es el conjunto que aparece en la NVD.

Este *score* se enmarca dentro de un *ranking* de tres posiciones, las vulnerabilidades consideradas como "*low*", cuya puntuación se sitúa entre 0.0 y 3.9; las vulnerabilidades

calificadas como "*medium*" con puntuación entre 4.0 y 6.9; y las vulnerabilidades clasificadas como "*high*" cuya valoración está entre 7.0 y 10.0 [17].

En caso de considerarse necesario, el *score* puede completarse con la asignación de valores a las métricas temporales y de entorno siguiéndose el mismo proceso que con las métricas de base. Las tres ecuaciones (ecuación de base, temporal y de entorno) pueden consultarse en [18]

Entrando con mayor detenimiento dentro del grupo de métricas utilizado por la NVD, la métrica de base se encarga de recoger las características estables tanto en tiempo como en entorno de una vulnerabilidad. Las métricas conocidas como *Access Vector*, *Access Complexity* y *Authentication* tratan sobre la forma de acceso a una vulnerabilidad y la existencia de las condiciones para ello. A partir de esta información se define el impacto de las tres otras métricas restantes, *Confidentiality*, *Integrity* y *Availability*.

A continuación, se explican cada una de las métricas que forman parte del grupo de métricas de base [18].

Access Vector

El *Access Vector* indica como es explotada la vulnerabilidad. Cuanto más remoto es un posible atacante a la aplicación, más alta es la puntuación de la vulnerabilidad. Los valores que puede tomar son los de la Tabla 4.

Valor	Descripción	Ejemplo
Local	El atacante tiene acceso físico al sistema o una cuenta local	Privilegios de escalado local (sudo)
Adjacent Network	Requiere acceso del atacante mediante *broadcast* o dominio de colisión	Redes locales: IP *subnet*, Bluetooth, segmento local Ethernet
Network	El *software* vulnerable está ligado a la pila de red	Desbordamiento del búfer del RPC

Tabla 4. Evaluación de los valores de la métrica Access Vector

Access Complexity

Este indicador mide la complejidad requerida por un atacante para obtener acceso al sistema considerando las condiciones de acceso existentes en él. Cuanto menor sea la complejidad de acceso requerida, mayor será su valoración. La Tabla 5 muestra los posibles valores del indicador.

Valor	Descripción	Ejemplo
High	Existen condiciones de acceso especializadas	Necesidad de privilegios elevados o sistemas adicionales. DNS *hijacking*
Medium	Las condiciones de acceso están algo especializadas	*Phishing* a través del que se modifica el browser de una página para mostrar enlaces falsos
Low	No existen condiciones de acceso especializadas	Configuraciones de acceso por defecto

Tabla 5. Evaluación de los valores de la métrica Access Complexity

Authentication

Esta métrica se encarga de recoger el número de veces que un atacante se debe autenticar en un sistema antes de poder explotar una vulnerabilidad del mismo por lo que no mide complejidad si no sólo si un atacante necesita credenciales para el acceso. Los posibles valores de *Authentication* son los de la Tabla 6. Cuanto menor sea el número de credenciales de autenticación, mayor será su puntuación en relación a esta métrica.

Valor	Descripción	Ejemplo
Multiple	Necesaria la autenticación dos o más veces	Credenciales para el acceso a un sistema operativo y también para el acceso a una aplicación del sistema
Single	Necesario autenticarse una vez	Credenciales para el acceso a un servidor de correo
None	No es necesario autenticarse	Sistemas sin ninguna autenticación

Tabla 6. Evaluación de los valores de la métrica *Authentication*

Confidentiality Impact

Esta métrica valora el impacto sobre la confidencialidad de un ataque a una vulnerabilidad que ha sido realizado con éxito. La confidencialidad está relacionada con la limitación de acceso a la información y su divulgación solamente a los usuarios con permisos para obtenerla. Así, cuanto mayor sea el impacto en la confidencialidad, mayor puntuación obtendrá. La Tabla 7 recoge los valores que puede tomar este indicador.

Valor	Descripción	Ejemplo
None	No hay impacto sobre la confidencialidad del sistema	No se accede a ninguna información relevante
Partial	No hay divulgación de información considerable	Se accede solamente a algunas tablas de una base de datos
Complete	Existe una divulgación total de la información	Se accede a ficheros, memoria, etc.

Tabla 7. Evaluación de los valores de la métrica *Confidentiality Impact*

Integrity Impact

El *Integrity Impact* mide la integridad de un sistema atacado. La integridad se refiere a la garantía sobre la fiabilidad y veracidad de la información. Los valores de este indicador son los que pueden observarse en la Tabla 8. A mayor impacto en la integridad del sistema, mayor puntuación del indicador.

Valor	Descripción
None	No hay impacto sobre la integridad del sistema
Partial	Posible la modificación de alguna información pero con efectos limitados
Complete	Pérdida completa de la protección de los datos

Tabla 8. Evaluación de los valores de la métrica *Integrity Impact*

Avaliability Impact

Se trata de un indicador sobre el impacto en la disponibilidad de un sistema tras un ataque realizado con éxito. La disponibilidad tiene que ver con la accesibilidad a los recursos de un sistema y está relacionada con ataques que consumen excesivo ancho de banda, ciclos largos de CPU o demasiado espacio en disco. Cuanto mayor sea su impacto en la disponibilidad de recursos mayor será su puntuación. En la Tabla 9 aparecen los posibles valores del *Avaliability Impact*.

Valor	Descripción
None	No hay impacto sobre la disponibilidad del sistema
Partial	Existen interrupciones o un rendimiento reducido del recurso
Complete	Caída completa del recurso

Tabla 9. Evaluación de los valores de la métrica *Availability Impact*

Capítulo 3. Descripción general del sistema

En este capítulo se proporciona una visión global de la aplicación presentándose su arquitectura a través de los elementos que la componen y sus funciones dentro de la aplicación.

Después se presenta su diseño, el cual se divide en cuatro módulos en función de su aportación dentro del sistema y se explica a grandes rasgos cada uno de ellos. De esta manera se pretende disponer de un punto de partida para que cada módulo, junto con su implementación, pueda ser desglosado y explicado en mayor profundidad más adelante.

Además, se especifican los requisitos funcionales y no funcionales de la aplicación y se presentan dos casos de uso de la misma.

3.1. Arquitectura

Como primer acercamiento al sistema implementado, se presenta su arquitectura a través de los elementos que la componen y sus relaciones. Algunos de los elementos son recursos externos, como el servidor NVD y el *XML* extraído de él, y el resto son elementos creados dentro del sistema implementado. La arquitectura del sistema puede observarse en la Figura 13.

Figura 13. Arquitectura general del sistema

El sistema se divide en dos bloques de seguridad diseñados para mitigar el riesgo existente: el gestor de vulnerabilidades, compuesto por el terminal con la aplicación móvil y su tabla de vulnerabilidades en la BBDD (Base de datos) de la aplicación, así como la BBDD NVD externa, el servidor Web intermedio y el protocolo para la comunicación con el mismo; y el bloque de seguridad por contexto, en el que intervienen el terminal con la aplicación y la tabla de contextos en su BBDD.

El usuario dispone, a través del dispositivo móvil, de la **aplicación** desarrollada. La aplicación consta de una interfaz para su interacción con ambos bloques de seguridad. Además, cuenta con una **BBDD** con dos tablas, una para almacenar las vulnerabilidades Android y otra para guardar la configuración de los contextos de seguridad. La BBDD se basa en SQLite ya que Android incorpora a través de sus APIs todas las herramientas necesarias para la creación y gestión de este tipo de BBDD.

Para la obtención de información que permita al usuario la detección de vulnerabilidades en el terminal móvil se plantearon dos posibilidades, la utilización de un NIDS (Network Intrusion Detection System) para Android o la NVD del NIST.

Para determinar la mejor opción, se analizaron cuatro NIDS diferentes (ver detalle en Anexo D), Crowdroid [19], Paranoid Android [20], Taintdroid [21] y Mockdroid [22]. Sin embargo los dos primeros no se tratan de una opción válida por no encontrarse disponibles en la Google Play Store para ser descargados e instalados en el dispositivo. Por otro lado, Taintdroid y Mockdroid a pesar de contar con un código fuente, precisan de un proceso de instalación laborioso que, en el caso de Mockdroid, es dependiente del hardware del dispositivo. Por tanto, finalmente se optó por la utilización de la información proporcionada por el repositorio público de vulnerabilidades del gobierno de los Estados Unidos conocido como NVD.

La NVD contiene información sobre las vulnerabilidades de productos software en diversos sistemas operativos. El acceso a dicha información se realiza a través de la descarga de un fichero *XML*. Sin embargo, para la extracción de las vulnerabilidades relacionadas exclusivamente con aplicaciones Android, es necesaria la creación e incorporación de un **servidor Web**, que actúa como intermediario entre el cliente y la NVD, ya que esta última no dispone de una API para hacer consultas directamente. Este servidor, además de discernir entre las vulnerabilidades que pertenecen a Android o a otro sistema, se encarga de obtener los principales campos de cada una de ellas, para proporcionar al dispositivo móvil solamente de la información más relevante, que será de la que disponga finalmente el usuario. Con esto conseguimos que la carga de procesado en el cliente sea más ligera, evitando así el consumo elevado de recursos (batería, procesador).

La comunicación entre el dispositivo, el servidor Web y la NVD se realiza a través de conexiones HTTP por tratarse de recursos en la red. Más concretamente, para la comunicación entre el terminal y el servidor Web, se establece un **protocolo**, a través del cual se fija el formato de los mensajes para el intercambio de información entre ellos, y mediante el cual, el dispositivo informa al servidor de la versión de su sistema operativo Android, que será del que quiera obtener las vulnerabilidades.

En cuanto a los datos para el establecimiento de contextos de seguridad, aunque existen aplicaciones para Android como AroundMe [23] y tripwolf [24] que disponen de datos

sobre el entorno en el que se encuentra el dispositivo, como restaurantes, estaciones de transporte público, hoteles u hospitales y que podrían analizarse para su utilización para la definición de contextos sin la intervención del usuario, en el sistema desarrollado se adquiere la información directamente desde el mismo dispositivo, mediante el uso de las APIs correspondientes y corresponde al usuario la creación de los contextos que quiera utilizar.

3.2. Funcionalidad

Como ya se ha presentado en el apartado anterior, el sistema se compone de dos bloques con aportaciones a la seguridad del dispositivo móvil basada en el riesgo existente, el **bloque de gestión de vulnerabilidades** y el **bloque de seguridad por contexto**. El primero se encarga de la obtención, detección y control de las vulnerabilidades presentes en aplicaciones del sistema operativo Android del dispositivo en cuestión, y el segundo bloque tiene como función la aplicación de configuraciones de seguridad en función del entorno en el que se encuentre el terminal. En mayor detalle, las funciones de cada bloque se explican a continuación.

La función principal del bloque de gestión de vulnerabilidades es dar a conocer al usuario las vulnerabilidades existentes en las diferentes aplicaciones disponibles para la versión del sistema operativo Android con el que cuenta el dispositivo y proporcionar un sistema para su control, mitigando así el riesgo de ataques.

La aplicación obtiene las vulnerabilidades Android publicadas en la NVD a través del servidor Web implementado para ello. Los pasos llevados a cabo para la obtención de vulnerabilidades, recogidos en la Figura 14, son los siguientes.

- **Obtención de vulnerabilidades**
    1. Desde el dispositivo, se realiza una petición al servidor Web, indicando su versión Android.
    2. El servidor Web recoge la petición y se descarga el fichero *XML* con las vulnerabilidades software de la NVD.
    3. El servidor Web procesa el fichero *XML* recibido y extrae un listado de vulnerabilidades Android correspondientes a la versión indicada.
    4. El servidor Web envía el listado de vulnerabilidades al dispositivo móvil.

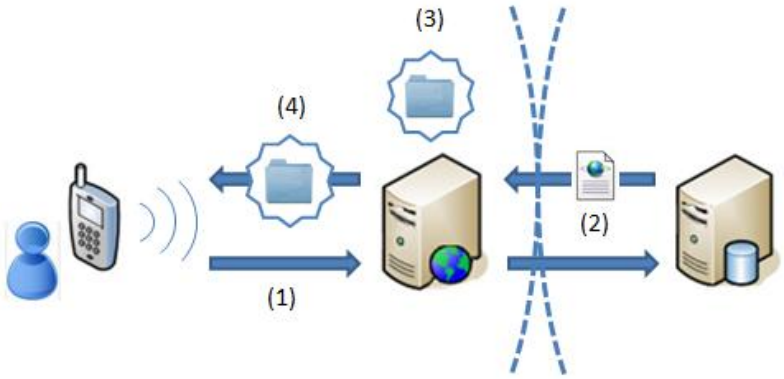

Figura 14. Procedimiento para la obtención de vulnerabilidades

Una vez con las vulnerabilidades en el dispositivo, éstas deben ser almacenadas en su tabla correspondiente de la base de datos de la aplicación con el fin de que los datos de cada una de ellas permanezcan guardados en el sistema del dispositivo en el que la aplicación esté instalada. Esto permite que las vulnerabilidades encontradas sean notificadas, consultadas y tratadas en cualquier momento por parte del usuario. El almacenamiento de vulnerabilidades se lleva a cabo siguiendo los pasos mostrados en la Figura 15 y explicados a continuación.

- **Almacenamiento de vulnerabilidades**
  1. Se analiza la lista de vulnerabilidades recibidas para comprobar cuales de los productos con alguna vulnerabilidad se encuentran instalados en el dispositivo.
  2. El dispositivo móvil comprueba si hay nuevas vulnerabilidades respecto a las que ya tiene almacenadas.

     2.1. Si la vulnerabilidad es nueva:

        2.1.1. Se almacena en la tabla de vulnerabilidades de la BBDD de la aplicación.

     2.2. Si la vulnerabilidad no es nueva:

        2.2.1. Se comprueba si se ha actualizado su información y en ese caso se almacena en la BBDD de la aplicación.

  3. En caso de existir nuevas vulnerabilidades o vulnerabilidades actualizadas, se notifica al usuario el número encontrado.

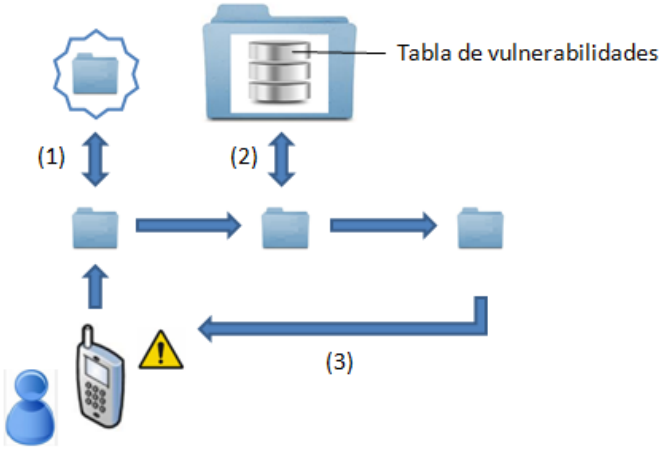

Figura 15. Procedimiento para el almacenamiento de vulnerabilidades

La consulta de las vulnerabilidades puede realizarse directamente a través de un histórico que permite obtener todas las almacenadas, mediante una búsqueda por fecha o palabra clave, o bien se pueden examinar solamente las vulnerabilidades pendientes de ser revisadas.

Al visualizar la información relativa a una vulnerabilidad, se proporciona, además de su descripción, un enlace con la solución propuesta en la página Web del fabricante. Mediante la consulta de este enlace, el usuario podrá aplicar las medidas recomendadas para la eliminación de la vulnerabilidad y, por tanto, reducir así el riesgo.

Asimismo, la aplicación permite llevar un control sobre las vulnerabilidades que ya hayan sido revisadas y las que quedan pendientes de serlo y proporciona un resumen con dicha información. La información se presenta siguiendo un código de colores, desde el rojo (alto), naranja (medio) y verde (bajo), lo que proporciona una noción visual del riesgo.

En cuanto al bloque para la seguridad por contexto, se presenta como una de las posibles aplicaciones prácticas de la conocida como seguridad por contexto. La seguridad por contexto, tal y como se explicó en apartado 2.1.1.6.1., pretende adaptar los parámetros de seguridad configurables del dispositivo al entorno en el que se encuentra. Estos contextos pueden definirse teniendo en cuenta diversos factores proporcionados por los sensores disponibles en el terminal.

Por tanto, la función de este módulo es gestionar la utilización de contextos de seguridad para su detección y la aplicación de sus medidas correspondientes para mitigar el riesgo.

Para la gestión de los contextos de seguridad, el sistema permite al usuario su definición, consulta, modificación y eliminación.

La detección del contexto en el que se encuentra el usuario propuesta en este proyecto se basa en la localización y en la hora del dispositivo. Respecto a las acciones a tomar en función del contexto, el sistema ofrece dos tipos de seguridad, la seguridad "Alta" y la seguridad "Baja", según determine el usuario en la creación del contexto. La seguridad "Alta" corresponde al uso de bloqueo de la pantalla del terminal mientras que la seguridad "Baja" elimina la utilización de bloqueo.

Los pasos para la aplicación de los contextos de seguridad se muestran en la Figura 16 y son presentados a continuación.

- **Aplicación de contextos de seguridad**

1. Se obtienen la información sobre las coordenadas y la hora actual del dispositivo.
2. Se consulta en la tabla de contextos de la BBDD de la aplicación si existe un contexto definido dentro de esos parámetros.
3. Se aplica el tipo de seguridad correspondiente.
   3.1. Si el contexto existe, se aplica el tipo de seguridad que indica el contexto de acuerdo al nivel de riesgo.
   3.2. Si el contexto no existe, se aplica un tipo de seguridad alta. Es decir, se asume por defecto que el riesgo es alto.

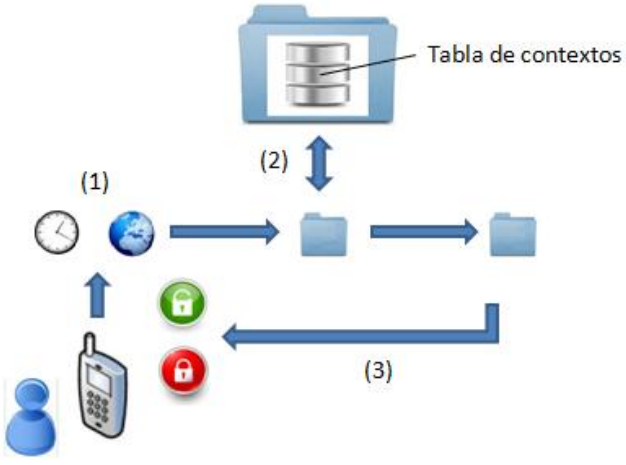

Figura 16. Procedimiento de la aplicación de seguridad por contexto

3.3. Diseño

Con el objetivo de simplificar su desarrollo y obtener un código modular, se ha realizado la división del sistema en cuatro módulos en función de su aportación al mismo: la interfaz de usuario, el módulo de gestión de vulnerabilidades, el módulo del servidor Web y el módulo de seguridad por contexto. Los tres primeros están implementados en el cliente mientras que el módulo 4 se encuentra desarrollado en el servidor. En la Figura 17 se presentan estos módulos y sus interacciones.

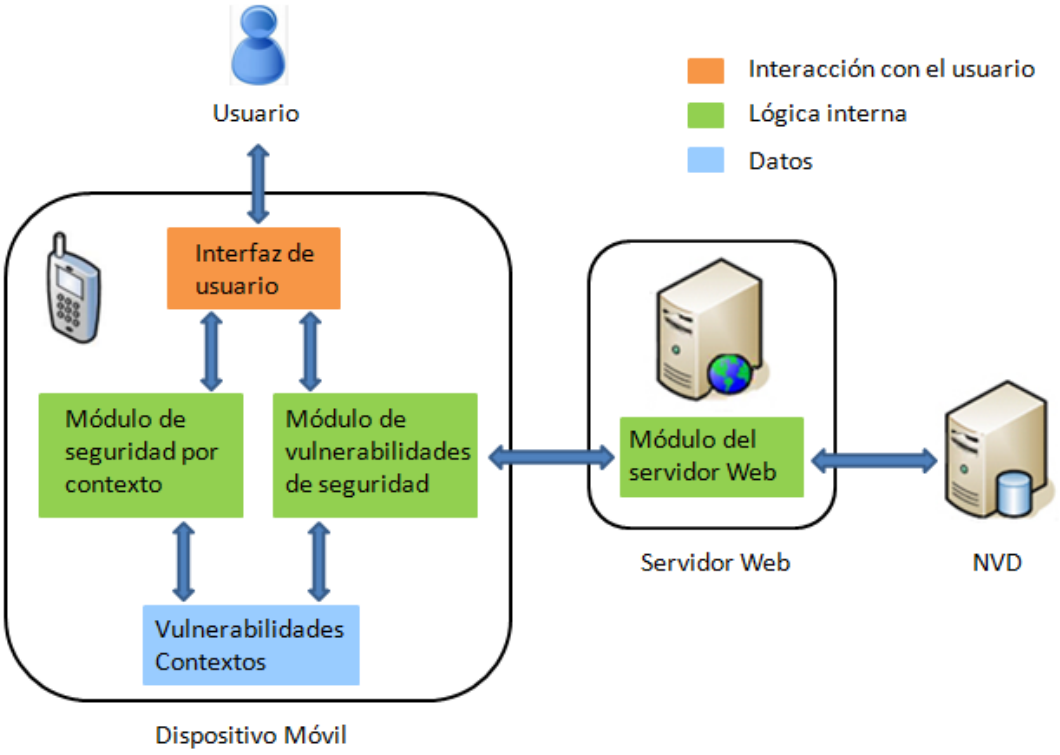

Figura 17. Diagrama de módulos de la aplicación

Cada uno de estos módulos, se presenta con mayor detalle en los sucesivos capítulos, aunque seguidamente se realiza una breve descripción para conocer, de modo general, el papel que desempeña cada uno de ellos dentro de la aplicación.

1. **Módulo de la interfaz de usuario.**

   La interfaz agrupa las distintas vistas que conforman la parte gráfica de la aplicación para la interacción del usuario con la lógica del sistema.

2. **Módulo de vulnerabilidades de seguridad.**

   Es el módulo encargado de gestionar las vulnerabilidades Android, permitiendo la búsqueda y consulta de las posibles vulnerabilidades presentes en el sistema. Además lleva el control sobre las vulnerabilidades que hayan sido revisadas, y

proporciona un resumen de las mismas. También se encarga de la notificación de las nuevas vulnerabilidades y de su almacenamiento.

3. **Módulo de seguridad por contexto.**

   Se encarga de la gestión de los contextos indicados por el usuario a través de la interfaz gráfica, es decir, lleva a cabo su creación, consulta, modificación y eliminación. Asimismo, recibe la información de localización y temporal del dispositivo para detectar si éste se encuentra dentro de un contexto de seguridad previamente definido y aplicar el tipo de seguridad correspondiente.

4. **Módulo del servidor Web.**

   Es el módulo con la misión de proporcionar la información sobre vulnerabilidades Android al dispositivo. Concretamente, el servidor Web actúa como intermediario entre el terminal y la NVD, encargándose de obtener y procesar los datos de las vulnerabilidades para su posterior envío al usuario.

Tal y como se recoge en la Figura 4, estos módulos pueden clasificarse en dos capas, la de interacción con el usuario y la de la lógica interna, las cuales involucran la interacción con la capa de datos. La capa de interacción con el usuario, se refiere exclusivamente la interfaz gráfica y por tanto recoge el módulo de la interfaz de usuario, mientras que la de lógica interna engloba los módulos que dan funcionalidad a la aplicación, es decir, el módulo de seguridad por contexto, el del servidor Web y el de gestión de vulnerabilidades. Por último, la capa de datos recoge las vulnerabilidades y los contextos almacenados en sus respectivas tablas de la BBDD y cuya explicación se incluye en los módulos pertenecientes la lógica interna.

3.4. Requisitos

Se procede a exponer los requisitos que debe cumplir la aplicación, obtenidos a partir de las especificaciones propuestas para su desarrollo y de las limitaciones con las que se ha contado.

Los requisitos del sistema se clasifican en requisitos de usuario y requisitos software y cada uno de ellos se muestra en una tabla que contiene su descripción y su prioridad o importancia en el desarrollo del sistema. Además la tabla incluye el campo origen en el caso de los requisitos de usuario y de software no funcionales, el cual es sustituido por el campo dependencia en los requisitos software funcionales. El origen se refiere a la

procedencia del requisito mientras que la dependencia indica una relación entre un requisito software funcional y un requisito de usuario de capacidad.

Asimismo, se utiliza un código basado en una abreviatura para la identificación de cada requisito. Éste se puede consultar en la Tabla 10.

<table>
<tr><th>Código</th><th colspan="3">Tipos de requisitos</th></tr>
<tr><td>RUC-nº</td><td colspan="2">Requisito de capacidad</td><td rowspan="2">Requisito de usuario</td></tr>
<tr><td>RUR-nº</td><td colspan="2">Requisito de restricción</td></tr>
<tr><td>RSF-nº</td><td colspan="2">Requisito funcional</td><td rowspan="3">Requisito software</td></tr>
<tr><td>RSNFI-nº</td><td>Requisito de interfaz</td><td rowspan="2">Requisito no funcional</td></tr>
<tr><td>RSNFO-nº</td><td>Requisito de operación</td></tr>
</table>

Tabla 10. Códigos de identificación de requisitos

3.4.1. Requisitos de usuario

Los requisitos de usuario son definidos por el cliente, el cual coincide con el desarrollador de la aplicación en este caso, y se subdividen en requisitos de capacidad, que son los referidos a las acciones que puede realizar el usuario, y en requisitos de restricción, que recogen las limitaciones impuestas por el cliente.

A continuación se muestran las tablas con los requisitos de capacidad y de restricción.

- **Requisitos de capacidad:**

RUC-01	**El usuario controla la actualización de las vulnerabilidades**		
Descripción	El usuario podrá decidir si activar o desactivar la actualización de la información sobre nuevas vulnerabilidades		
Origen	Cliente	**Prioridad**	Alta

Tabla 11. RUC-01 Controlar la actualización de vulnerabilidades

RUC-02	**El usuario visualiza los datos de una vulnerabilidad**		
Descripción	El usuario podrá visualizar los datos de una vulnerabilidad		
Origen	Cliente	**Prioridad**	Alta

Tabla 12. RUC-02 Visualizar una vulnerabilidad

RUC-03	El usuario determina si una vulnerabilidad ha sido revisada		
Descripción	El usuario indicará en cada una de las vulnerabilidades si ésta ha sido solucionada en el dispositivo		
Origen	Cliente	**Prioridad**	Alta

Tabla 13. RUC-03 Controlar la revisión de vulnerabilidades

RUC-04	El usuario visualiza un resumen de vulnerabilidades		
Descripción	El usuario podrá ver un resumen relacionado con la seguridad del dispositivo, en cuanto a vulnerabilidades en aplicaciones Android se refiere		
Origen	Cliente	**Prioridad**	Alta

Tabla 14. RUC-04 Visualizar un resumen de vulnerabilidades

RUC-05	El usuario controla la utilización de contextos de seguridad		
Descripción	El usuario podrá elegir si se utilizan los contextos de seguridad		
Origen	Cliente	**Prioridad**	Alta

Tabla 15. RUC-05 Controlar el uso de contextos

RUC-06	El usuario crea un contexto de seguridad		
Descripción	El usuario creará contextos de seguridad		
Origen	Cliente	**Prioridad**	Alta

Tabla 16. RUC-06 Crear un contexto

RUC-07	El usuario visualiza los datos de un contexto		
Descripción	El usuario podrá visualizar los datos de un contexto previamente creado		
Origen	Cliente	**Prioridad**	Alta

Tabla 17. RUC-07 Visualizar un contexto

RUC-08	El usuario modifica los datos de un contexto		
Descripción	El usuario podrá modificar los datos de un contexto previamente definido		
Origen	Cliente	**Prioridad**	Alta

Tabla 18. RUC-08 Modificar un contexto

RUC-09	El usuario elimina un contexto		
Descripción	El usuario podrá eliminar un contexto previamente definido		
Origen	Cliente	**Prioridad**	Alta

Tabla 19. RUC-09 Eliminar un contexto

- **Requisitos de restricción:**

RUR-01	**La interfaz es intuitiva**		
Descripción	La interfaz de la aplicación muestra *toasts*[5] y *hints*[6] para hacer más intuitiva su utilización		
Origen	Cliente	**Prioridad**	Alta

Tabla 20. RUR-01 Interfaz intuitiva

RUR-02	**La interfaz está en castellano**		
Descripción	La interfaz de la aplicación está en castellano		
Origen	Cliente	**Prioridad**	Alta

Tabla 21. RUR-02 Interfaz en castellano

RUR-03	**El sistema está en vertical**		
Descripción	La orientación de la aplicación es vertical		
Origen	Cliente	**Prioridad**	Alta

Tabla 22. RUR-03 Interfaz en vertical

RUR-04	**El sistema utiliza el protocolo HTTP**		
Descripción	Las comunicaciones en el sistema utilizan el protocolo HTTP		
Origen	Cliente	**Prioridad**	Alta

Tabla 23. RUR-04 Protocolo HTTP

RUR-05	**El sistema garantiza su funcionamiento a partir de Android 2.2.x Froyo**		
Descripción	La mínima versión del sistema operativo Android que garantiza el correcto funcionamiento de la aplicación es Android 2.2.x Froyo		
Origen	Cliente	**Prioridad**	Alta

Tabla 24. RUR-05 Funcionamiento a partir de Android 2.2.x

3.4.2. Requisitos software

Los requisitos software recogen el comportamiento del sistema y se dividen en requisitos funcionales, que forman la base que determinará el funcionamiento de la aplicación, y no funcionales, relacionados con las restricciones impuestas por la implementación del sistema.

Seguidamente se presentan las tablas con los requisitos funcionales y no funcionales del sistema.

[5] El término *toast* se refiere al mensaje mostrado al usuario durante unos segundos al producirse algún cambio del que se debe avisar

[6] El término *hint* se refiere al mensaje mostrado en el receptor de texto cuando éste se encuentra vacío.

- **Requisitos funcionales**:

RSF-01	**Activación y desactivación de la actualización de vulnerabilidades**		
Descripción	El sistema permite habilitar o deshabilitar la recepción de la información sobre nuevas vulnerabilidades		
Dependencia	RUC-01	**Prioridad**	Alta

Tabla 25. RSF-01Actualización de vulnerabilidades

RSF-02	**Obtención automática de vulnerabilidades**		
Descripción	La aplicación obtendrá las nuevas vulnerabilidades de la versión Android del dispositivo que hayan sido publicadas en la NVD		
Dependencia		**Prioridad**	Alta

Tabla 26. RSF-02 Obtención de vulnerabilidades

RSF-03	**Notificación de las nuevas vulnerabilidades**		
Descripción	La aplicación notificará el número de nuevas vulnerabilidades encontradas		
Dependencia		**Prioridad**	Alta

Tabla 27. RSF-03 Notificación de vulnerabilidades

RSF-04	**Búsqueda de vulnerabilidades por fecha**		
Descripción	El sistema permite buscar entre las vulnerabilidades almacenadas en la aplicación a partir de su fecha de publicación		
Dependencia	RUC-02	**Prioridad**	Baja

Tabla 28. RSF-04 Búsqueda de vulnerabilidades por fecha

RSF-05	**Búsqueda de vulnerabilidades por palabras clave**		
Descripción	El sistema permite buscar entre las vulnerabilidades almacenadas en la aplicación en función de una o varias palabras clave contenidas en su sumario o descripción		
Dependencia	RUC-02	**Prioridad**	Alta

Tabla 29. RSF-05 Búsqueda de vulnerabilidades por palabra clave

RSF-06	**Búsqueda de vulnerabilidades por fecha y por palabra clave**		
Descripción	El sistema permite combinar ambas búsquedas de vulnerabilidades almacenadas en el mismo		
Dependencia	RUC-02	**Prioridad**	Baja

Tabla 30. RSF-06 Búsqueda de vulnerabilidades por fecha y por palabra clave

RSF-07	Histórico de vulnerabilidades		
Descripción	El sistema permite consultar un histórico con todas las vulnerabilidades		
Dependencia	RUC-02	**Prioridad**	Alta

Tabla 31. RSF-07 Histórico de vulnerabilidades

RSF-08	Acceso a la solución de una vulnerabilidad		
Descripción	La aplicación permitirá el acceso a la solución propuesta por el fabricante para cada una de las vulnerabilidades		
Dependencia		**Prioridad**	Alta

Tabla 32. RSF-08 Solución a una vulnerabilidad

RSF-09	Control de la revisión de una vulnerabilidad		
Descripción	El sistema permite dejar constancia de si se ha revisado cada una de las vulnerabilidades en el mismo		
Dependencia	RUC-03	**Prioridad**	Alta

Tabla 33. RSF-09 Revisión de una vulnerabilidad

RSF-10	Resumen de vulnerabilidades y seguridad del sistema		
Descripción	El sistema mostrará un resumen relacionado con el número de vulnerabilidades pendientes de ser revisadas		
Dependencia	RUC-04	**Prioridad**	Alta

Tabla 34. RSF-10 Ver resumen de vulnerabilidades

RSF-11	Consulta de vulnerabilidades pendientes de revisión		
Descripción	El sistema permitirá consultar las vulnerabilidades pendientes de revisión de forma directa		
Dependencia		**Prioridad**	Alta

Tabla 35. RSF-11 Ver vulnerabilidades pendientes de revisión

RSF-12	Activación y detección del uso de contextos		
Descripción	La aplicación permitirá activar y desactivar la utilización de contextos de seguridad		
Dependencia	RUC-05	**Prioridad**	Alta

Tabla 36. RSF-12 Activación y desactivación del uso de contextos

RSF-13	Definición de un contexto		
Descripción	El sistema permitirá la definición de contextos de seguridad		
Dependencia	RUC-06	**Prioridad**	Alta

Tabla 37. RSF-13 Creación de un contexto

RSF-14	Consulta de un contexto		
Descripción	El sistema permitirá la consulta de los contextos de seguridad definidos y sus datos		
Dependencia	RUC-07	**Prioridad**	Alta

Tabla 38. RSF-14 Consulta de un contexto

RSF-15	Modificación de un contexto		
Descripción	El sistema permitirá la modificación de un contexto de seguridad		
Dependencia	RUC-08	**Prioridad**	Alta

Tabla 39. RSF-15 Modificación de un contexto

RSF-16	Eliminación de un contexto		
Descripción	El sistema permitirá la eliminación de un contexto de seguridad		
Dependencia	RUC-09	**Prioridad**	Alta

Tabla 40. RSF-16 Eliminación de un contexto

RSF-17	Detección de un contexto		
Descripción	La aplicación debe detectar cuando se produce un cambio de contexto y si el dispositivo se encuentra dentro de un contexto conocido o no		
Dependencia		**Prioridad**	Alta

Tabla 41. RSF-17 Detección de un contexto

RSF-18	Aplicación de la configuración de seguridad		
Descripción	El sistema aplicará la configuración de seguridad correspondiente al contexto en el que se encuentra el dispositivo		
Dependencia		**Prioridad**	Alta

Tabla 42. RSF-18 Aplicación de seguridad

RSF-19	El sistema corre en segundo plano		
Descripción	El sistema funcionará en segundo plano		
Origen	Cliente	**Prioridad**	Alta

Tabla 43. RSF-19 Segundo plano

- **Requisitos no funcionales:**

Los requisitos no funcionales están divididos a su vez en requisitos de interfaz, por los que se determina como debe ser ésta, en requisitos de comunicación, que fijan los estándares de comunicación empleados y en los requisitos de operación, relacionados con las restricciones y elementos necesarios para el funcionamiento del sistema.

Al tratarse el cliente y el desarrollador de este proyecto de la misma persona, los requisitos no funcionales coinciden en su mayoría con los requisitos de restricción. Seguidamente se presentan reagrupados en categorías.

- **Requisitos de interfaz:**

RSNFI-01	**La interfaz es intuitiva**		
Descripción	La interfaz de la aplicación muestra *toasts* y *hints* para hacer más intuitiva su utilización		
Origen	Cliente	**Prioridad**	Alta

Tabla 44. RSNFI-01 Interfaz intuitiva

RSNFI-02	**La interfaz está en castellano**		
Descripción	La interfaz de la aplicación está en castellano		
Origen	Cliente	**Prioridad**	Alta

Tabla 45. RSNFI-02 Interfaz en castellano

RSNFI-03	**El sistema está en vertical**		
Descripción	La orientación de la aplicación es vertical		
Origen	Cliente	**Prioridad**	Alta

Tabla 46. RSNFI-03 Interfaz en vertical

- **Requisitos de comunicación:**

RSNFC-01	**El sistema utiliza el protocolo HTTP**		
Descripción	Las comunicaciones en el sistema utilizan el protocolo HTTP		
Origen	Cliente	**Prioridad**	Alta

Tabla 47. RSNFC-01 Protocolo HTTP

- **Requisitos de operación:**

RSNFO-01	**El sistema dispone de GPS**		
Descripción	El dispositivo necesita de un GPS para la obtención de su localización		
Origen	Cliente	**Prioridad**	Alta

Tabla 48. RSNFO-01 Disponer de GPS

RSNFO-02	**El sistema dispone de conexión a Internet**		
Descripción	El dispositivo necesita de conexión a una red de datos para la transferencia de la información sobre vulnerabilidades		
Origen	Cliente	**Prioridad**	Alta

Tabla 49. RSNFO-02 Disponer de conexión a Internet

RSNFO-03	El sistema garantiza su funcionamiento a partir de Android 2.2.x Froyo		
Descripción	La mínima versión del sistema operativo Android que garantiza el correcto funcionamiento de la aplicación es Android 2.2.x Froyo		
Origen	Cliente	**Prioridad**	Alta

Tabla 50. RSNFO-03 Funcionamiento a partir de Android 2.2.x

3.5. Casos de uso

Los casos de uso presentan el sistema a través de su utilización por parte del usuario. El usuario de la aplicación podrá realizar diversas acciones. En la Figura 18 se muestran esas acciones, las cuales conforman dos casos de uso, el caso 1, referente al bloque de gestión de vulnerabilidades, y el caso 2, asociado al bloque de seguridad por contexto.

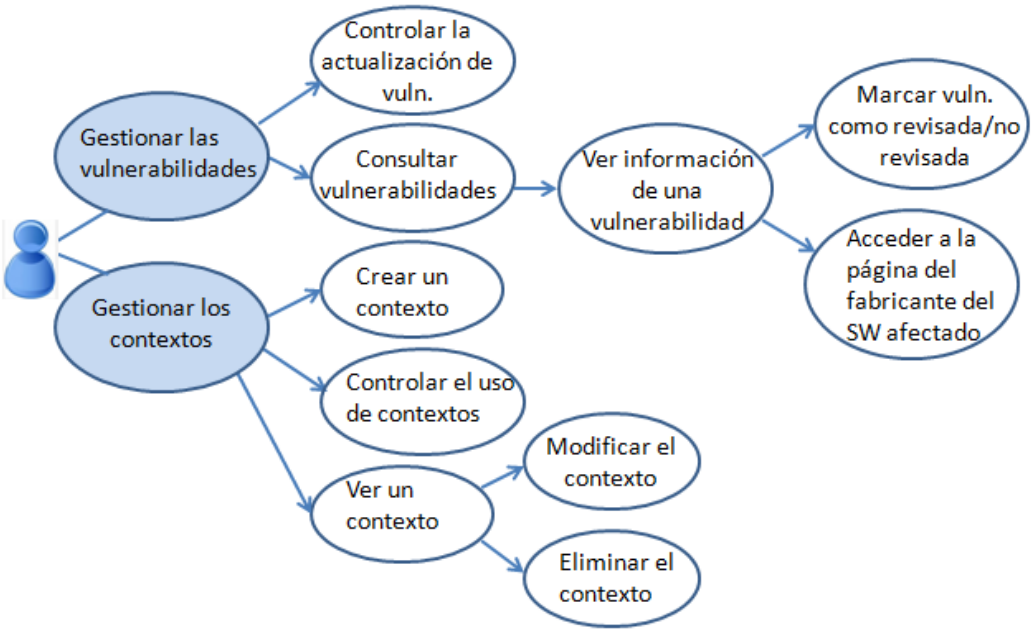

Figura 18. Diagrama que recoge los casos de uso y su flujo

Los dos casos de uso se recogen en sus respectivas tablas, mostradas a continuación. En cada tabla se especifican los actores del caso, la dependencia con los requisitos de usuario y la descripción y el flujo que sigue el mismo.

CU-01	Gestionar las vulnerabilidades
Actores	Usuario
Dependencia	RUC-01, RUC-02, RUC-03 y RUC-04
Descripción	Gestionar la información sobre vulnerabilidades para su recepción, consulta y solución
Flujo	1. El usuario activa la recepción de nueva información sobre vulnerabilidades y su actualización, la cual puede también detener, y consulta las vulnerabilidades notificadas o ya almacenadas

	2. El usuario visualiza la información de una vulnerabilidad 3. El usuario accede a la solución propuesta por el fabricante y tras tomar las acciones necesarias, marca la vulnerabilidad como revisada

Tabla 51. Gestión de vulnerabilidades

CU-02	Gestionar los contextos
Actores	Usuario
Dependencia	RUC-05, RUC-06, RUC-07, RUC-08 y RUC-09
Descripción	Gestionar la activación del uso de contextos de seguridad, así como su creación, modificación y eliminación
Flujo	1. El usuario crea varios contextos de seguridad 2. A partir de que se haya definido algún contexto, el usuario activa la utilización de contextos, la cual puede detener si lo considera oportuno 3. El usuario visualiza los datos de un contexto y los modifica o elimina el contexto

Tabla 52. Gestión de contextos

Capítulo 4. Implementación del sistema

En este capítulo se explica en detalle todo lo relacionado con la implementación del sistema, mostrándose las clases, actividades y servicios que forman parte de él, así como sus métodos.

Por tanto, se explica el desarrollo de la aplicación, el cual está dividido en dos partes, la relacionada con la implementación de la interacción con el usuario y la de la implementación de la lógica interna del sistema. Dentro de la primera parte se explica la estructura de la interfaz de usuario así como los elementos que forman sus vistas y en la segunda parte se pasa a explicar cómo está desarrollada cada una de las tareas que aportan funcionalidad a la aplicación.

4.1. Introducción

Una vez conocido el diseño del sistema, se pasa a explicar su implementación. Ésta se ha dividido en dos partes, la implementación de la interacción con el usuario y la de la lógica interna.

Siguiendo este criterio, las clases, actividades y servicios que forman parte del sistema desempeñan una función relacionada puramente con la interfaz, con la lógica interna o en algún caso con ambas

Con el fin de tener una idea general de las clases, actividades y servicios desarrollados para la construcción de la aplicación y de sus relaciones, se presenta el esquema de la Figura 19. Estos componentes y sus métodos son explicados en mayor detalle en los sucesivos apartados.

4.2. Interacción con el usuario

Para el uso de la aplicación es necesaria una interfaz que permita la interacción con el usuario. La interfaz gráfica está formada por vistas. Cada una de estas vistas se refiere a las diferentes pantallas que aparecen a lo largo del uso de la aplicación. El usuario accede a las funcionalidades de la misma a través de los elementos de estas vistas. Las acciones que podrá realizar están relacionadas con la activación o desactivación de las funcionalidades de la aplicación y la consulta y gestión de vulnerabilidades y contextos de seguridad.

4.2.1. Interfaz de usuario

En Android, cada vista se corresponde con una actividad, dentro de la cual se pueden lanzar las pantallas y dar funcionalidad a los elementos de la misma, además de contener otros métodos que puedan estar más relacionados con la lógica del sistema tal y como se verá más adelante en el apartado 4.3.

Las actividades y los métodos que intervienen en la creación de la interfaz de usuario se recogen en la Figura 20. Algunas de estas actividades sólo forman parte de la interfaz de usuario, como es el caso de *MostrarVulnActivity* o *VisorVulnActivity*, mientras que otras también intervendrán en la lógica del sistema, como *VulnActivity* o *ContextActivity*.

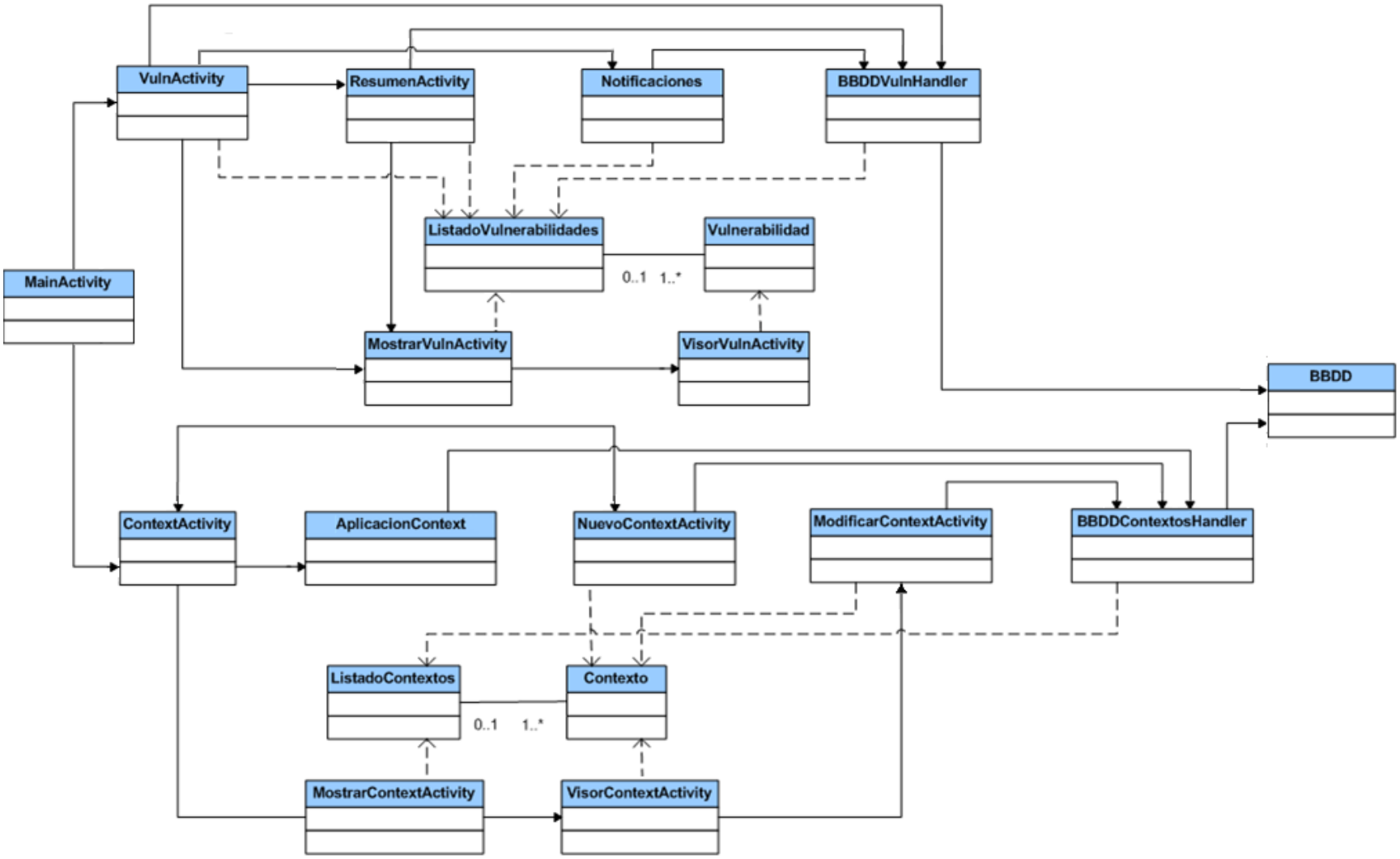

Figura 19. Clases, actividades y servicios de la aplicación en el dispositivo móvil

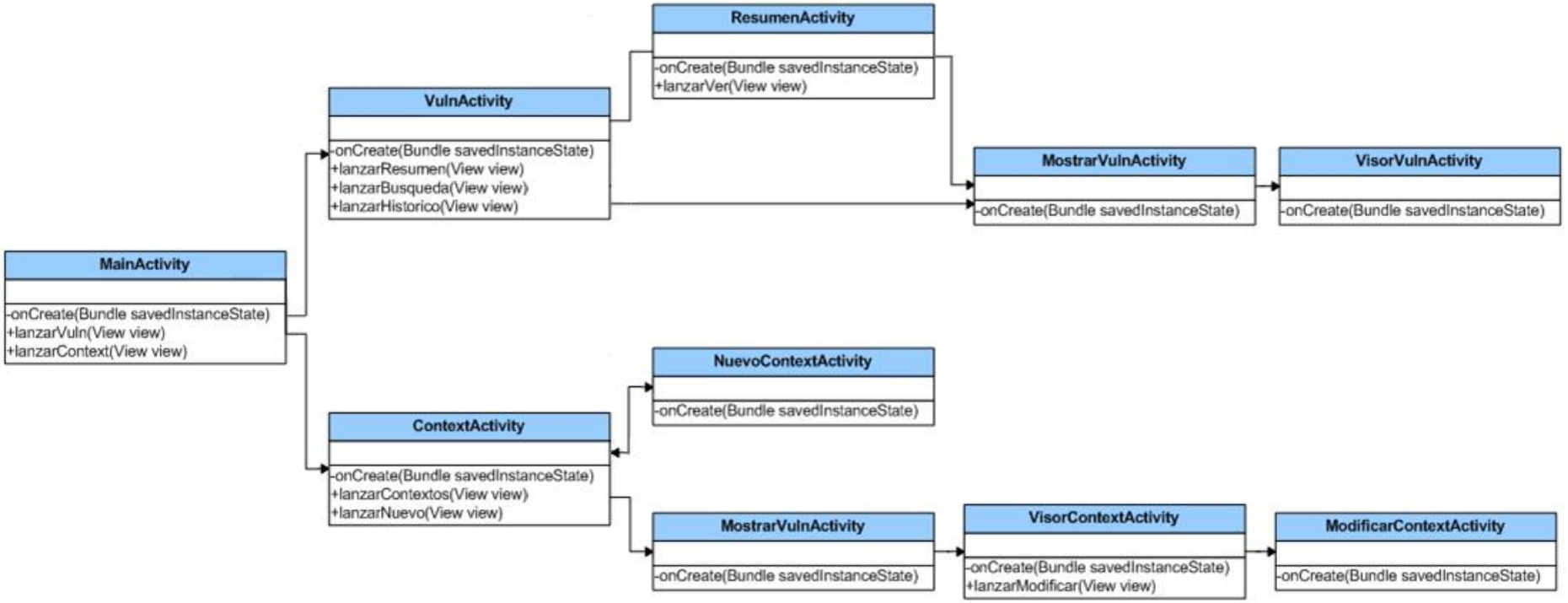

Figura 20. Actividades que componen la interfaz de usuario

Cada actividad es la encargada de lanzar la creación de su vista asociada a través del método *onCreate(Bundle savedInstanceState)*. Para el cambio entre unas pantallas y otras, algunas de las actividades cuentan también con los métodos *lanzar(View view)*, encargados de pasar a la vista que corresponda según la acción tomada. En otras actividades, en cambio, no existe el método lanzar bien porque se tratan de una vista final que no da paso a ninguna otra, o bien porque sus *layouts* son dinámicos, es decir, no tienen una estructura fija sino que son creados desde el código.

4.2.1.1. Estructura de las vistas

Las vistas que componen la interfaz de usuario se basan en diferentes tipos de recursos para su construcción. Para la creación de las vistas de la aplicación se han utilizado tres tipos recursos, *layouts*, *values* y *drawables*, los cuales se explican seguidamente.

Layouts

Los ficheros *XML* almacenados dentro de la ruta */res/layout* de la aplicación contienen la definición de las diferentes pantallas que forman la interfaz. Estos ficheros permiten posicionar los elementos gráficos dentro de la pantalla del dispositivo además de la definición de los diferentes parámetros visuales (tamaño, color, estilo) de cada uno de ellos.

Values

En */res/values* se encuentran los ficheros para la definición de las constantes que pueden ser utilizadas como parámetros de los elementos de un *layout*. Los ficheros de valores por defecto son el fichero de cadenas y el de estilos. El fichero *strings.xml* contiene cadenas de caracteres fijas mientras que el fichero *styles.xml* recoge varios estilos definidos por defecto para su aplicación a los elementos de una vista.

Drawables

Los ficheros localizados en */res/drawable* contienen recursos gráficos que pueden ser utilizados como imágenes directamente dentro de la pantalla. Se pueden definir diferentes recursos en función de la resolución y densidad de la pantalla del terminal de forma que se definen varias subcarpetas para ello, desde */drawable-ldpi* para una densidad baja hasta */drawable-xhdp* para una densidad muy alta.

La interfaz gráfica del sistema se compone de varios ficheros de estos tipos.

En primer lugar, la aplicación está formada por diez *layouts* distintos, es decir, que cuenta con diez diferentes vistas. El fichero *activity_main* es la vista principal a partir de la que se accede al resto de pantallas. Los *layouts* que componen la aplicación y los cambios entre ellos quedan reflejados en la Figura 21.

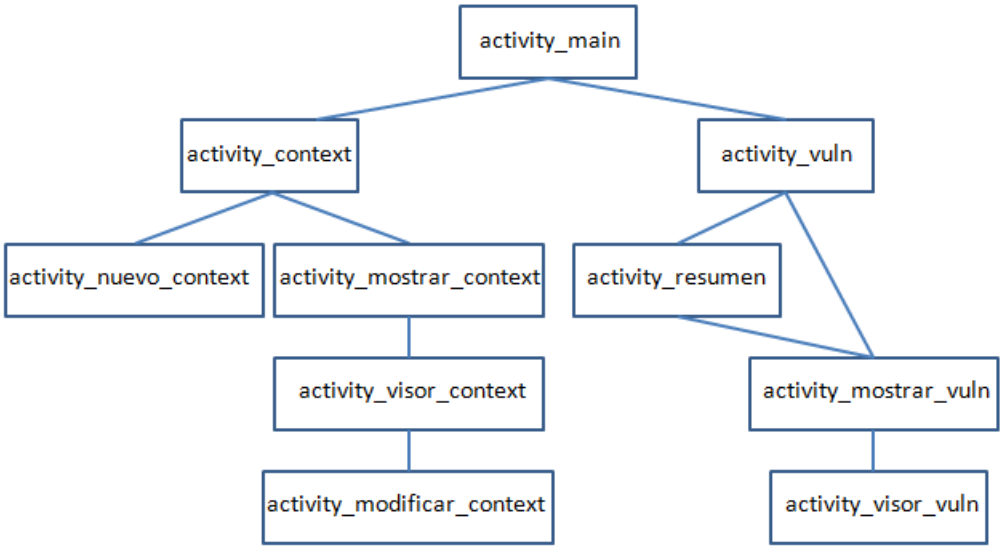

Figura 21. Estructura de los Layouts de la aplicación

En cuanto a los ficheros que definen constantes, además de con un fichero *strings.xml*, y un fichero *styles.xml*, la aplicación cuenta con un fichero *color.xml* en el que se definen colores diferentes a los existentes por defecto con el fin de colorear textos.

Finalmente, se dispone de varios recursos de tipo *drawable* para una densidad de pantalla alta, consistentes en imágenes intuitivas y decorativas con extensión *.png* utilizadas en algunas de las vistas.

4.2.1.2. Descripción de las vistas y acciones asociadas

Se procede a la detallar cada una de las diez vistas de la aplicación mediante la descripción de los *layouts*, uno por cada actividad del sistema, que componen su estructura, así como las acciones asociadas a sus elementos. Estas vistas y sus relaciones se muestran en la Figura 22.

Vista del *layout activity_main*

Es la vista que contiene el menú de inicio de la aplicación. A través de él, se accede a la parte de seguridad por contexto o a la de vulnerabilidades. La vista se compone de los siguientes elementos:

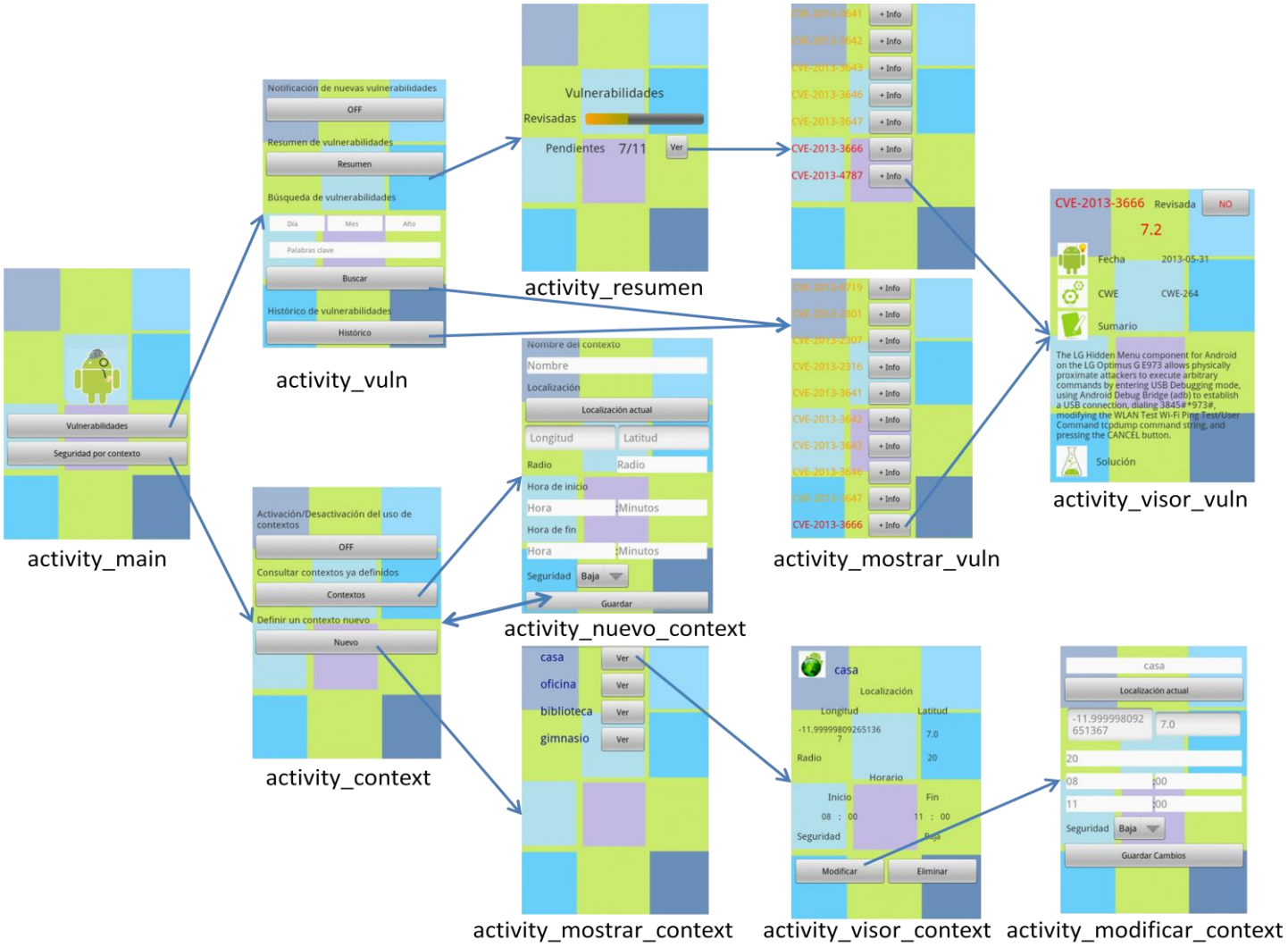

Figura 22. Vistas de la aplicación y sus relaciones

- Botón *Vulnerabilidades*: proporciona acceso al layout *activity_vuln.*
- Botón *Seguridad por contexto*: proporciona acceso al layout *activity_context.*

Vista del layout *activity_vuln*

El *layout activity_vuln* se corresponde a la vista con los elementos para la consulta de vulnerabilidades de seguridad mediante distintos métodos. Además, proporciona el control para la activación y desactivación de la notificación de nuevas vulnerabilidades. Otros de sus elementos son:

- Botón *ON/OFF*: permite activar o desactivar el lanzamiento periódico de peticiones de nuevas vulnerabilidades.
- Botón *Resumen*: proporciona acceso al *layout activity_resumen* para consultar las vulnerabilidades pendientes de ser revisadas.
- Receptores de texto *Día*, *Mes* y *Año*: recogen la fecha a partir de la cual se desea encontrar vulnerabilidades.
- Receptor de texto *Palabras clave*: recoge una palabra o una lista de palabras relacionadas con la vulnerabilidad que se desea encontrar.
- Botón *Buscar*: proporciona acceso al *layout activity_mostrar_vuln* para observar las vulnerabilidades que cumplan con los criterios de búsqueda dados.
- Botón *Histórico*: proporciona acceso al *layout activity_mostrar_vuln* para observar todas las vulnerabilidades almacenadas a modo de histórico en el terminal.

Vista del *layout activity_resumen*

Esta vista muestra un resumen sobre las vulnerabilidades en el sistema y ofrece la posibilidad de su consulta. Los elementos que la componen se presentan a continuación:

- Barra *Revisadas*: muestra el porcentaje de vulnerabilidades revisadas respecto al número total de vulnerabilidades mediante la variación del relleno y color del contenido de la barra.
- Visor de texto *Pendientes*: recoge el número de vulnerabilidades pendientes de ser revisadas frente al total de vulnerabilidades registradas en el sistema.
- Botón *Ver*: proporciona acceso al *layout activity_mostrar_vuln* para la consulta exclusivamente de las vulnerabilidades pendientes de revisión.

Vista del *layout activity_mostrar_vuln*

Se trata de una vista con un *layout* dinámico, es decir, que está formado por un número variable de elementos que aparecen en pantalla según el número de contextos de seguridad que se encuentren almacenados en el dispositivo. Estos elementos son visores de texto con los nombres de cada uno de los identificadores CVE de cada una de las vulnerabilidades y junto a cada una de ellas se encuentra el siguiente elemento:

- Botón *Ver*: proporciona acceso al *layout activity_visor_vuln*.

Vista del *layout activity_visor_vuln*

En ella se muestran los datos de una vulnerabilidad a través de varios visores de texto. Los datos mostrados son el identificador de la vulnerabilidad, su fecha de publicación, su código CWE, su sumario y la página web del fabricante con la solución que propone. Además, la vista contiene algunos datos más específicos sobre la vulnerabilidad. La vista cuenta con otro elemento más, el botón Resuelto:

- Botón *Revisada*: cambia su valor entre "SI" y "NO", mediante el cual el usuario indica si ha revisado la vulnerabilidad en su dispositivo o no.

Vista del *layout activity_context*

En *activity_context* se define la vista con los elementos para la creación y consulta de contextos de seguridad, además de la activación o desactivación de su uso. Sus principales elementos son:

- Botón *ON/OFF*: permite la activación o desactivación del uso de contextos de seguridad.
- Botón *Contextos*: proporciona acceso al *layout activity_mostrar_context*.
- Botón *Nuevo*: proporciona acceso al *layout activity_nuevo_context*.

Vista del *layout activity_nuevo_context*

El *layout activity_nuevo_context* recoge los datos proporcionados por el usuario para la creación de un nuevo contexto de seguridad.

- Receptor de texto con *hint Nombre*: recoge el nombre con el que se identifica el contexto de seguridad que se pretende crear.
- Botón *Localización actual*: recoge la posición actual en la que se localiza el dispositivo y muestra dicha información en los receptores de texto con *hint Longitud* y *Latitud*.

- Receptores de texto con *hint Longitud* y *Latitud*: muestran las coordenadas en longitud y latitud respectivamente de la localización actual del terminal.
- Receptor de texto con *hint Radio*: recoge el radio que tendrá el contexto.
- Receptores de texto con *hint Hora* y *Minutos* de inicio y de fin: recogen el horario de inicio y el horario de fin del contexto de seguridad.
- Selector *Seguridad*: despliega un menú que permite elegir entre dos tipos de seguridad, "Baja" y "Alta".
- Botón *Guardar*: guarda el contexto creado y retorna a la vista con *layout activity_context*.

Vista del *layout activity_mostrar_context*

De igual manera que con el *layout activity_mostrar_vuln*, se trata de una vista con un *layout* dinámico compuesto por un número variable de elementos según el número de vulnerabilidades de seguridad almacenadas en el dispositivo. Estos elementos son visores de texto con los nombres de cada una de las vulnerabilidades y junto a cada una de ellas aparece el siguiente elemento:

- Botón *Ver*: proporciona acceso al *layou*t *activity_visor_context*.

Vista del *layout activity_visor_context*

Es la vista que muestra por pantalla todos los datos de un contexto previamente definido a través de varios visores de texto. Estos datos son el nombre del contexto, su localización en longitud y latitud, su radio, su horario de duración y su tipo de seguridad. Asimismo, la vista cuenta con dos botones:

- Botón *Modificar*: proporciona acceso al *layou*t *activity_modificar_context*.
- Botón *Eliminar*: permite borrar un contexto. Tras pulsarlo se pide la confirmación de la acción.

Vista del *layout activity_modificar_context*

Se trata de una vista similar a la proporcionada por el *layout activity_nuevo_context* pero cuyos receptores de texto contienen *hints* con los datos correspondientes al contexto que se pretende modificar a modo de guía.

- Botón *Guardar Cambios*: guarda las modificaciones realizadas en el contexto cy retorna a la vista con *layout activity_context*.

En la interacción con la mayoría de estas vistas, se van mostrando notas informativas a través de *toasts*, los cuales confirman las acciones realizadas o informan sobre posibles errores en la información introducida por el usuario.

4.3. Lógica interna

Para dar funcionalidad a la aplicación, se precisa de diversas acciones internas. Estas acciones se recogen en la lógica interna del sistema, que se divide en tres módulos: el módulo de Vulnerabilidades de Seguridad, el del Servidor Web, y el módulo de Seguridad por Contexto.

Para cada uno de estos módulos se explican, a continuación, las clases utilizadas, los métodos implementados y las relaciones entre ellos.

4.3.1. Vulnerabilidades de seguridad

El módulo de vulnerabilidades de seguridad se encarga de la adquisición de la información sobre vulnerabilidades y de su almacenamiento y gestión. Para ello, cuenta con varias clases, actividades y un servicio, mostradas en la Figura 23.

Las clases implementadas, *Vulnerabilidad* y *ListadoVulnerabilidades*, permiten definir la estructura de una vulnerabilidad y el manejo de listados de vulnerabilidades. También son clases *BBDD* y *BBDDVulnHandler*, a través de las que se crea y maneja la tabla de vulnerabilidades de la BBDD de la aplicación.

En cuanto a las actividades, este módulo reparte sus tareas entre *VulnActivity* y *ResumenActivity*. Asimismo, requiere del servicio *Notificaciones*, implementado para mantener la funcionalidad del módulo cuando el usuario sale de la aplicación.

Más concretamente, la funcionalidad del módulo se divide entre la parte encargada de obtener la información sobre nuevas vulnerabilidades y su notificación al usuario, y la parte que lleva a cabo las acciones indicadas por el usuario a través de la interfaz para la gestión de vulnerabilidades. Cada una de estas funciones es explicada en detalle en los siguientes apartados.

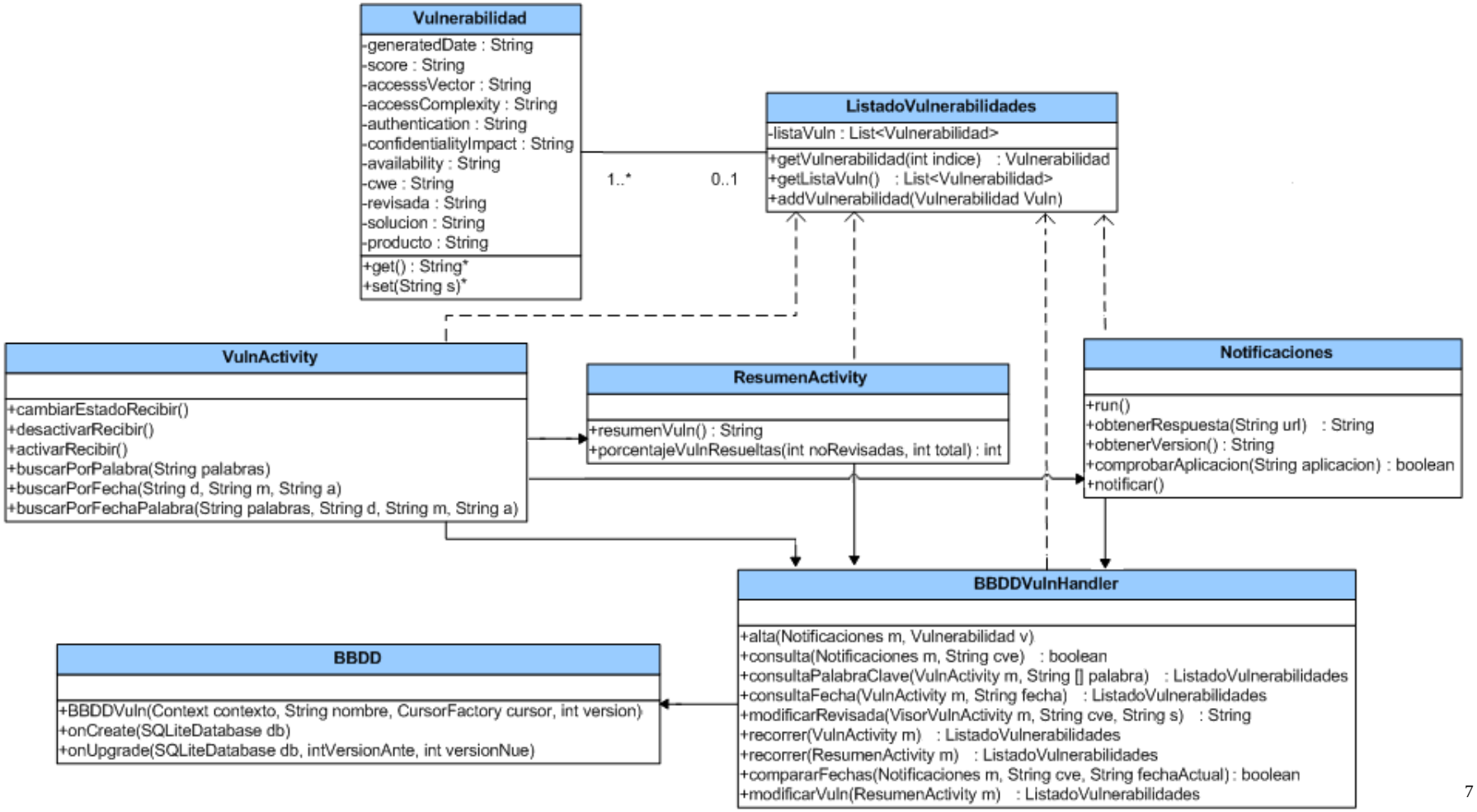

Figura 23. Clases, actividades y servicio del módulo de vulnerabilidades de seguridad

[7] En la clase Vulnerabilidad los métodos *get()* y *set(String s)* se refieren a los métodos get y set correspondientes a cada uno de sus atributos

4.3.1.1. Estructura de una vulnerabilidad

En primer lugar se determinan los datos que componen la estructura de una vulnerabilidad. Esta estructura está definida a partir de los recursos y estándares de identificación, clasificación y medición utilizados en la NVD. Concretamente, cada vulnerabilidad almacenada en el sistema está formada por los campos presentados en la Tabla 53.

Campos	Descripción
generatedDate	Fecha de la última modificación de la vulnerabilidad
score	Cada uno de ellos se corresponde con las métricas de base explicadas en el apartado 2.2.1.2.4. Proporcionan información adicional sobre la vulnerabilidad
accessVector	
acessComplexity	
authenticathion	
confidentialityImpact	
integrityImpact	
availability	
cwe	Identificador CWE
summary	Resumen con la descripción de la vulnerabilidad
cve	Identificador de la vulnerabilidad
solucion	Enlace a la página Web con la solución del fabricante a la vulnerabilidad
producto	Nombre del producto con la vulnerabilidad
revisada	Indicador de si la vulnerabilidad ha sido revisada y solucionada en el sistema

Tabla 53. Campos de una vulnerabilidad

El usuario tiene acceso a todos estos campos al visualizar el contenido de una vulnerabilidad, proporcionándosele una información detallada sobre la misma. Además, algunos de ellos, como *cve* o *revisada*, son utilizados a su vez para la gestión de las vulnerabilidades en el sistema y otros como *generatedDate* o *producto* son empleados para determinar si una vulnerabilidad ha actualizado sus datos o en caso de ser nueva, si el producto se encuentra instalado en el dispositivo, tal y como se explicará más adelante.

Para el manejo de las vulnerabilidades, se crean listas utilizando la clase *ListadoVulnerabilidades*.

4.3.1.2. Lógica para la obtención de los datos

En este apartado se explica cómo está implementada la obtención de los datos de las vulnerabilidades dentro del dispositivo.

El procedimiento general para la obtención de las vulnerabilidades Android es el recogido en el diagrama de flujo de la Figura 24.

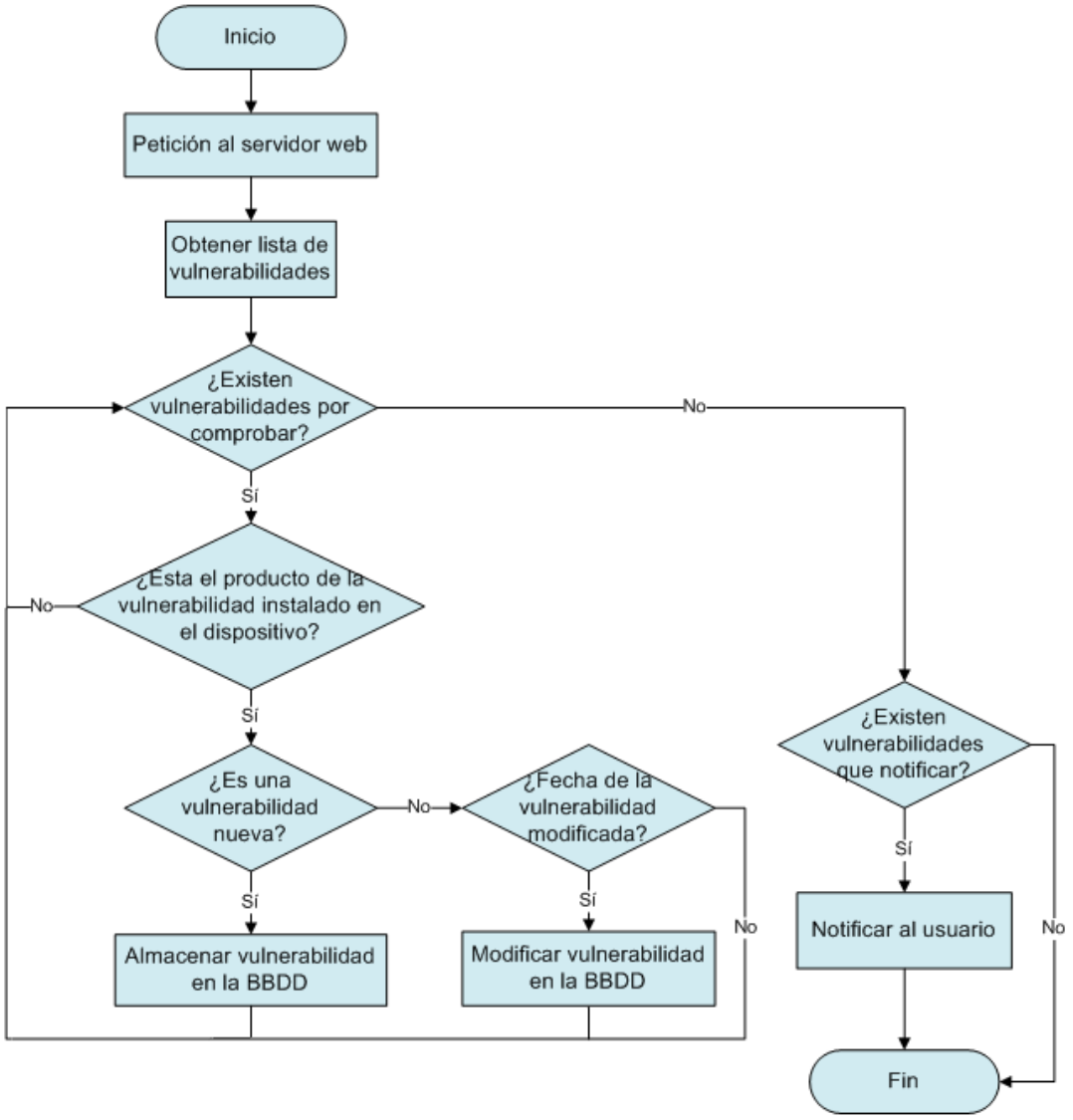

Figura 24. Diagrama de flujo para la obtención de vulnerabilidades

Las peticiones de vulnerabilidades al servidor Web se lanzan en segundo plano. Para ello, se utiliza la el servicio *Notificaciones*, el cual se encarga de la comunicación con el

servidor, ejecutada en su método *run()* y el cuál es controlado desde la actividad *VulnActivity* mediante los métodos *activarRecibir()* y *desactivarRecibir()*.

La información sobre vulnerabilidades *software* es actualizada diariamente en la NVD, tal y como puede comprobarse consultado la fecha de actualización junto al enlace del fichero *nvdcve-2.0-2013.xml* en la página con los ficheros de alimentación de la misma [25]. Al descargar este fichero no se indica su fecha de expiración, por lo que se utiliza un período de 24 horas entre peticiones de nuevas vulnerabilidades, proporcionándose de esta manera, su mayor seguimiento posible. En las peticiones al servidor se incluye la versión del sistema operativo del mismo dispositivo Android, la cual se obtiene a través del valor del parámetro *VERSION.RELEASE* de la clase *Build*.

Dentro del método *run()* también se llevan a cabo las comprobaciones para analizar cada vulnerabilidad recibida. A partir de la respuesta recibida del servidor Web, se obtienen las vulnerabilidades solicitadas y se comprueba si el producto al que se refiere cada una de ellas se encuentra instalado en el dispositivo mediante el método *comprobarAplicacion(String aplicacion)* que a su vez emplea el método *getPackageManager()* de la clase *Context* de la API de Android para conocer que aplicaciones instaladas. En caso de estarlo, se determina si la vulnerabilidad es nueva a través del método *consulta(Notificaciones m, Vulnerabilidad v)* del manejador *BBDDVulnHandler*, que informa sobre si la tabla de vulnerabilidades de la BBDD de la aplicación contiene una vulnerabilidad previamente almacenada con el identificador de la vulnerabilidad en cuestión. Si la vulnerabilidad es nueva, se almacena en la tabla mediante el método *alta(Notificaciones m, Vulnerabilidad v)* y en caso de no ser nueva, se comprueba si su fecha de generación ha sido modificada a través de *compararFechas(Notificaciones m, String cve, String fechaActual)*, ambos métodos pertenecientes también al manejador. Si lo ha sido, es decir, si la información de la vulnerabilidad ha sido actualizada, se modifica su entrada en la tabla utilizando el método *modificarVuln(Notificaciones m, Vulnerabilidad v)*.

Cuando existan nuevas vulnerabilidades o vulnerabilidades cuyos datos hayan sido actualizados, se notifica al usuario a través del método *notificar()*.Tras la notificación al usuario, la ejecución termina hasta pasado el período entre peticiones.

4.3.1.3. Lógica para la gestión de vulnerabilidades

La aplicación permite realizar varias acciones sobre las vulnerabilidades almacenadas en ella, las cuales se implementan en la actividad *VulnActivity*.

Las acciones llevadas a cabo para la gestión de las vulnerabilidades se engloban en mostrar un resumen sobre las vulnerabilidades en el sistema, permitir al usuario indicar si una vulnerabilidad ha sido revisada o no, realizar búsquedas sobre ellas y mantener un histórico de las mismas.

Resumen de vulnerabilidades

El resumen sobre vulnerabilidades presenta al usuario información sobre el número de vulnerabilidades revisadas frente al total de vulnerabilidades encontradas en el sistema. El diagrama de flujo del proceso se muestra seguidamente (Figura 25).

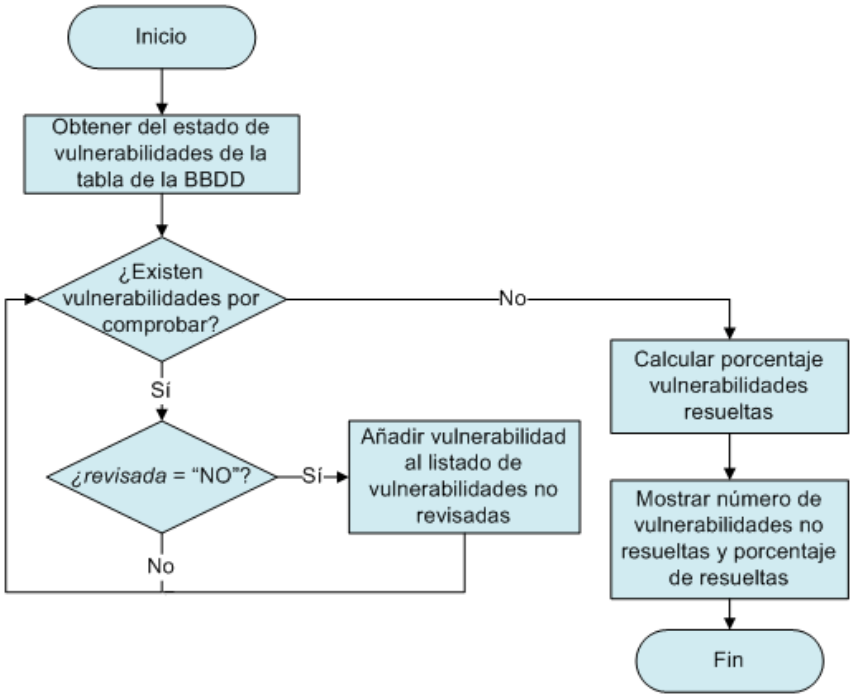

Figura 25. Diagrama de flujo para el resumen de vulnerabilidades

Para determinar si una vulnerabilidad ha sido revisada se emplea el método *resumenVuln()* de la actividad *ResumenActivity*. A través de éste, se consultan las vulnerabilidades almacenadas en la tabla de vulnerabilidades de la BBDD de la aplicación utilizando el método *recorrer(ResumenActivity m)* del manejador *BBDDVulnHandler* y se comprueba el estado del atributo *revisada* en cada caso. Si revisada indica que no lo está, es decir, es igual al valor "NO", se mantiene en una lista de vulnerabilidades no revisadas que servirá para proporcionar el resumen al cliente.

Al terminar de comprobar todas las vulnerabilidades almacenadas, se calcula el porcentaje de vulnerabilidades resueltas mediante el método *porcentajeVulnResvisadas(int noRevisadas, int total)* de *ResumenActivity*.

Por otro lado, la funcionalidad para que el usuario pueda indicar si una vulnerabilidad ha sido revisada se implementa en la clase *VisorVulnActivity* (englobada en la parte de interacción con el usuario) ya que al tratarse de la actividad que muestra la información de la vulnerabilidad, también recoge la indicación por parte del usuario de que una vulnerabilidad ha sido revisada y refleja este cambio en la entrada de la misma en la tabla de vulnerabilidades de la BBDD mediante el método *modificarRevisada(VisorVulnActivity m, String cve, String s)* del manejador.

Búsqueda de vulnerabilidades

En cuanto a la búsqueda de vulnerabilidades, ésta se realiza desde la actividad *VulnActivity*. Existen tres métodos diferentes en función de los parámetros indicados para ello. El primer de ellos es por fecha, es decir, que devuelve las vulnerabilidades con fecha de publicación igual o posterior a la señalada por el usuario. El segundo busca vulnerabilidades por palabra o palabras clave presentes en el sumario de la vulnerabilidad y el tercero combina los criterios de búsqueda del primer y segundo método para obtener vulnerabilidades por fecha y por palabras clave. Los pasos que forman este procedimiento se recogen en la Figura 26.

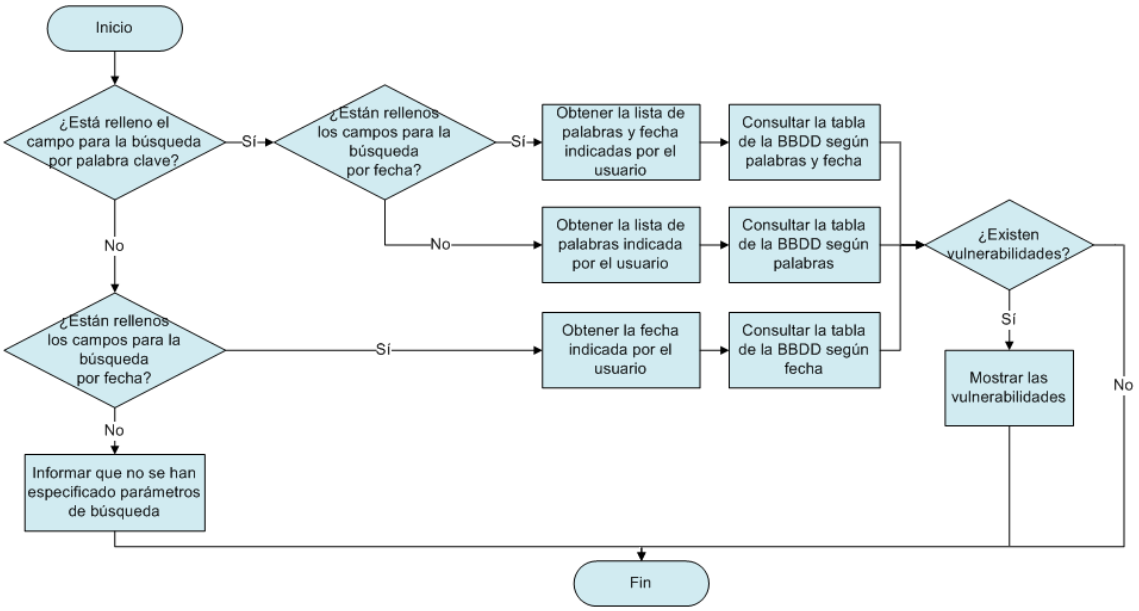

Figura 26. Diagrama de flujo para la búsqueda de vulnerabilidades

En cualquier búsqueda, por tanto, se comprueban primeramente cuáles son los datos que han sido indicados por el usuario con el objetivo de determinar el tipo de búsqueda que desea realizar mediante comprobaciones sobre los campos de la interfaz *activity_vuln* que hayan sido rellenados por el usuario.

Después, se obtienen los datos proporcionados para la búsqueda y se lleva a cabo la consulta de las vulnerabilidades almacenadas en la BBDD que cumplan con ellos a través de los métodos *buscarPorPalabra(String palabras)*, *buscarPorFecha(String d, String m, String a)*, y *buscarPorFechaPalabra(String palabras, String d, String m, String a)* de *VulnActivity*, que interactúan con los métodos *consultaPalabraClave(VulnActivity m, String [] palabra)* y *consultaFecha(VulnActivity m, String fecha)* del manejador *BBDDVulnHandler* para obtener el listado con las vulnerabilidades que coinciden con la búsqueda.

Finalmente, si se han encontrado vulnerabilidades, se muestran a través de la vista *VisorVulnActivity*.

Histórico de vulnerabilidades

Por último, el histórico de vulnerabilidades permite mostrar todas las vulnerabilidades de seguridad almacenadas en la BBDD. Se trata de una función con un diagrama de flujo sencillo, presentado en la Figura 27.

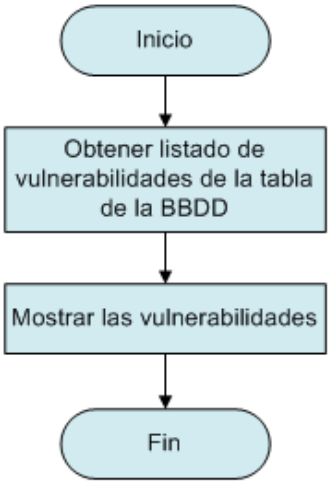

Figura 27. Diagrama de flujo para el histórico de vulnerabilidades

El método utilizado para obtener el histórico de vulnerabilidades es *recorrer(VulnnActivity m)* de *BBDDVulnHandler*, el cual es llamado desde la actividad

VulnActivity. Una vez se obtiene la lista de vulnerabilidades, se muestra a través de *VisorVulnActivity*.

4.3.2. Servidor Web

El servidor Web es el encargado de proveer de información sobre vulnerabilidades a la aplicación mediante su interacción con el módulo de vulnerabilidades de seguridad.

Concretamente, el servidor Web actúa como elemento intermedio entre el cliente y la NVD para el procesado de la información que requiere el dispositivo móvil. La decisión de utilizar un servidor intermedio se debe a dos cuestiones. La primera de ellas es la de simplificar la aplicación en el cliente evitando realizar el procesamiento de los datos en el mismo y la segunda y más importante, es la limitación que supone trabajar con ficheros de peso considerable en terminales móviles, cuyo tamaño de memoria es reducido en comparación con otros dispositivos electrónicos, luego con el empleo del servidor Web se evita la descarga del fichero de vulnerabilidades en el dispositivo, que puede llegar a superar los 100 MB.

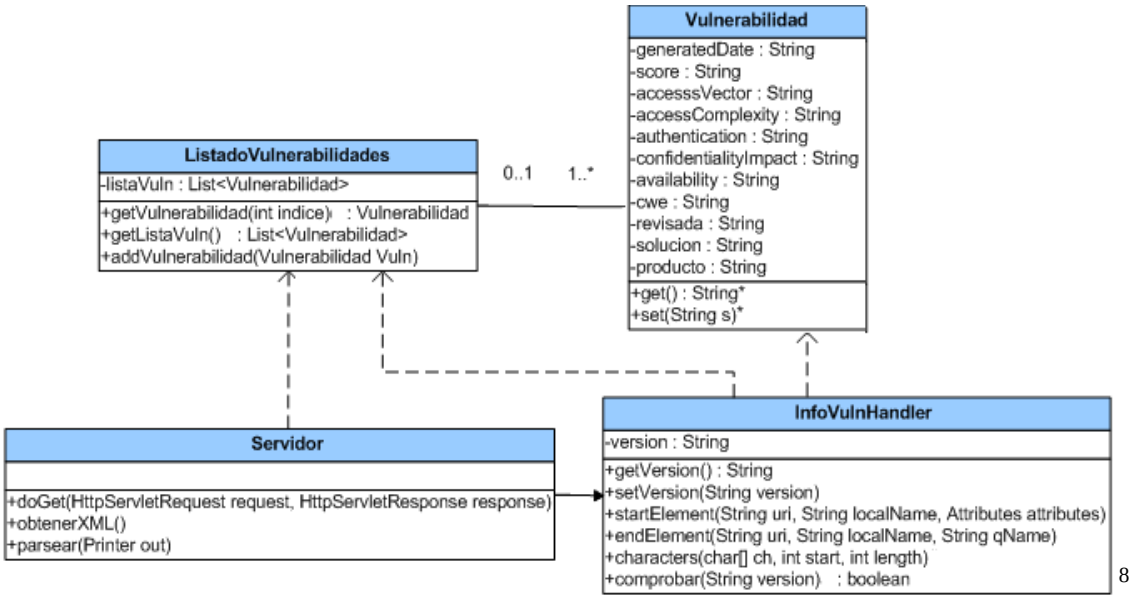

Figura 28. Clases que intervienen en el servidor Web

Como servidor Web se utiliza Apache Tomcat ya que se trata de un servidor sencillo desarrollado en Java, como el resto de la aplicación. En su contenedor Web se ejecuta el

[8] En la clase Vulnerabilidad los métodos *get()* y *set(String s)* se refieren a los métodos get y set correspondientes a cada uno de sus atributos

Servlet *Servidor* encargado de la comunicación con la aplicación en el cliente y con la NVD. El procesado del fichero procedente de la NVD se realiza a través de un parseador implementado en la clase *InfoVulnHandler*. Además, en este módulo se requiere de las clases *Vulnerabilidades* y *ListadoVulnerabilidades*. Estas clases y sus relaciones se muestran en la Figura 28.

4.3.2.1. Comunicación con otros elementos

El servidor Web se comunica con la aplicación cliente para el intercambio de vulnerabilidades Android y con la NVD para la descarga del fichero de vulnerabilidades de seguridad. Estas transferencias de información están basadas en un modelo de solicitud-respuesta y utilizan el protocolo HTTP al tratarse de una comunicación entre elementos situados en la Web o con conexión a ella. Esta comunicación y los elementos que intervienen en ella quedan reflejados en la Figura 29.

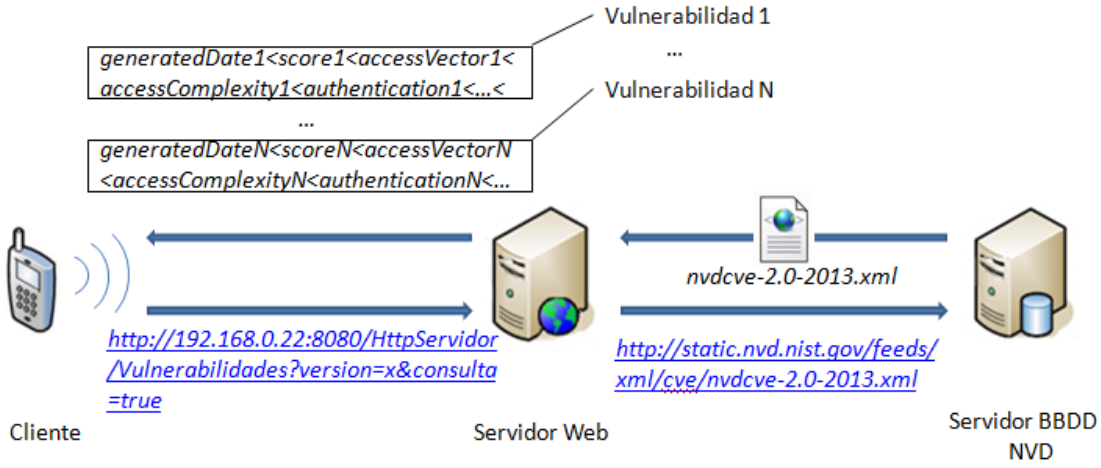

Figura 29. Comunicación del servidor Web con el cliente móvil y la NVD

El proceso de comunicación del servidor con el resto de elementos está implementado en el Servlet denominado *Servidor* el cual utiliza el API Servlet. Este Servlet se encuentra corriendo en un puerto de una dirección IP de forma permanente a la espera de peticiones. A continuación se explican las conexiones realizadas entre el servidor y el cliente y entra el servidor y la NVD para la transferencia de la información.

Para el intercambio de información entre el cliente y el servidor Web, se define un protocolo de comunicación. La estructura de la petición desde el terminal móvil es la siguiente:

http://192.168.0.22:8080/HttpServidor/Vulnerabilidades?version=x&consulta=true

Figura 30. Formato de la petición del cliente al servidor Web

Las peticiones se componen de la dirección IP en la que se encuentra corriendo el servidor Web seguidas del nombre de la aplicación Web, en este caso *HttpServidor* y del patrón de la URL asociado al Servlet, *Vulnerabilidades*. Asimismo, incluye los parámetros *versión* y *consulta*. El parámetro *versión* contiene el número de versión del sistema operativo Android del dispositivo cliente del que se desean obtener las vulnerabilidades y el parámetro *consulta* controla que el servidor sólo lance peticiones a la NVD cuando se reciba la petición desde el cliente.

En cuanto a la respuesta proporcionada por el servidor Web, el servidor crea una cadena compuesta de las vulnerabilidades a enviar al cliente. Los campos que componen cada una de las vulnerabilidades están separados por el carácter '<', escogido como separador tras comprobarse que no aparece dentro de la información de los campos implicados. Asimismo, este separador será de ayuda para la recomposición de las vulnerabilidades en la aplicación cliente. Un ejemplo de la traza de respuesta del servidor Web al dispositivo móvil es el mostrado a continuación:

```
2013-04-22T00:00:00.000-04:00<1.9<LOCAL<MEDIUM<NONE<NONE<NONE<PARTI
AL<CWE-20<The avast! Mobile Security application before 2.0.4400 fo
r Android allows attackers to cause a denial of service (applicatio
n crash) via a crafted application that sends an intent to com.avas
t.android.mobilesecurity.app.scanner.DeleteFileActivity with zero a
rguments.<CVE-2013-0122<NO<http://www.kb.cert.org/vuls/id/131263<av
as%21t_avast%21_mobile_security<2013-03-06T00:07:11.163-05:00<10.0<
NETWORK<LOW<NONE<COMPLETE<COMPLETE<COMPLETE<CWE-119<Buffer overflow
 in Adobe Flash Player before 10.3.183.50 and 11.x before 11.5.502.
146 on Windows and Mac OS X, before 10.3.183.50 and 11.x before 11.
2.202.261 on Linux, before 11.1.111.31 on Android 2.x and 3.x, and
before 11.1.115.36 on Android 4.x; Adobe AIR before 3.5.0.1060; and
 Adobe AIR SDK before 3.5.0.1060 allows attackers to execute arbitr
ary code via unspecified vectors.<CVE-2013-0630<NO<http://lists.ope
nsuse.org/opensuse-security-announce/2013-01/msg00003.html<adobe_ad
obe_air_sdk<2013-03-07T00:03:47.260-05:00<9.3<NETWORK<MEDIUM<NONE<C
```

Figura 31. Extracto de una respuesta del servidor Web al cliente

En esta respuesta puede observarse el orden establecido para el envío de los datos que componen cada vulnerabilidad, siendo su estructura general la que sigue:

generatedDate<score<accessVector<acessComplexity<authenticathion<
confidentialityImpact<integrityImpact<availability<cwe<summary<cve<
revisada<solucion<producto

Figura 32. Formato de la respuesta del servidor Web

En cuanto a la comunicación entre el servidor Web y la NVD, en ese caso, el servidor se convierte en cliente para descargarse el fichero *nvdcve-2.0-2013.xml* de la NVD El *nvdcve-2.0-2013.xml* se trata de un fichero alimenta a la NVD del NIST con todas las vulnerabilidades de seguridad publicadas a lo largo del año 2013 por los fabricantes de productos *software* y sus datos pueden ser solicitados en la dirección *http://static.nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2013.xml* [25].

4.3.2.2. Procesado de los datos

La parte de procesado de los datos es la encargada de localizar y extraer la información necesaria del fichero de vulnerabilidades de seguridad. Esta tarea se lleva a cabo en la clase *InfoVulnHandler* mediante el uso de un parseador XML. Para la realización del parseador, se ha utilizado el API SAX que permite la creación de un parseador genérico de XMLs, el cual se ha adaptado a la estructura del fichero *nvdcve-2.0-2013.xml.*

```
  </entry>
- <entry id="CVE-2013-0122">
   - <vuln:vulnerable-configuration id="http://nvd.nist.gov/">
      - <cpe-lang:logical-test operator="OR" negate="false">
            <cpe-lang:fact-ref name="cpe:/a:avas%21t:avast%21_mobile_security:2.0.4304:-:%7E%7E%7Eandroid%7E%7E"/>
        </cpe-lang:logical-test>
    </vuln:vulnerable-configuration>
   - <vuln:vulnerable-software-list>
        <vuln:product>cpe:/a:avas%21t:avast%21_mobile_security:2.0.4304:-:%7E%7E%7Eandroid%7E%7E</vuln:product>
    </vuln:vulnerable-software-list>
    <vuln:cve-id>CVE-2013-0122</vuln:cve-id>
    <vuln:published-datetime>2013-04-21T23:27:12.987-04:00</vuln:published-datetime>
    <vuln:last-modified-datetime>2013-04-22T00:00:00.000-04:00</vuln:last-modified-datetime>
   - <vuln:cvss>
      - <cvss:base_metrics>
            <cvss:score>1.9</cvss:score>
            <cvss:access-vector>LOCAL</cvss:access-vector>
            <cvss:access-complexity>MEDIUM</cvss:access-complexity>
            <cvss:authentication>NONE</cvss:authentication>
            <cvss:confidentiality-impact>NONE</cvss:confidentiality-impact>
            <cvss:integrity-impact>NONE</cvss:integrity-impact>
            <cvss:availability-impact>PARTIAL</cvss:availability-impact>
            <cvss:source>http://nvd.nist.gov</cvss:source>
            <cvss:generated-on-datetime>2013-04-22T10:14:00.000-04:00</cvss:generated-on-datetime>
        </cvss:base_metrics>
    </vuln:cvss>
    <vuln:cwe id="CWE-20"/>
   - <vuln:references xml:lang="en" reference_type="UNKNOWN">
        <vuln:source>CERT-VN</vuln:source>
        <vuln:reference xml:lang="en" href="http://www.kb.cert.org/vuls/id/131263">VU#131263</vuln:reference>
    </vuln:references>
    <vuln:summary>The avast! Mobile Security application before 2.0.4400 for Android allows attackers to cause a denial of service
        (application crash) via a crafted application that sends an intent to com.avast.android.mobilesecurity.app.scanner.
        DeleteFileActivity with zero arguments.</vuln:summary>
</entry>
```

Figura 33. Estructura de una vulnerabilidad en el fichero *nvdcve-2.0-2013.xml*

La implementación del parseador consiste en la definición de etiquetas fijas a detectar dentro del XML. A través de ellas se puede determinar el comienzo y el final de un

elemento mediante los métodos *startElement(String uri, String localName, String qName, Attributes attributes)* y *endElement(String uri, String localName, String qName)* y obtener su contenido utilizando el método *characters(char []ch, int start, int length).*

Para la obtención de las vulnerabilidades, el parseador recorre el fichero y cuando una de las etiquetas definidas es detectada, almacena su valor, tras lo que continúa buscando nuevas etiquetas hasta obtener la última relacionada con la vulnerabilidad, correspondiente al sumario de la misma. En ese caso se comprueba si la versión Android de la petición del cliente se corresponde con la del sumario de la vulnerabilidad y si lo hace, se crea un objeto de la clase Vulnerabilidad con los campos almacenados hasta entonces y por último, ésta se añade a un listado de vulnerabilidades a partir del cual se creará la respuesta al cliente. De igual manera se continúa examinando el fichero en busca de nuevas vulnerabilidades con la versión Android solicitada hasta su fin.

Como se puede comprobar, al no existir ningún campo en la vulnerabilidad que la clasifique según su sistema operativo, se hace necesario buscar esa información en el sumario de la misma. Esto ralentiza el descarte de vulnerabilidades en este proceso, sin embargo, no supone una limitación temporal en el sistema ya que las peticiones de vulnerabilidades se lanzan de forma transparente al usuario.

4.3.3. Seguridad por contexto

El módulo de seguridad por contexto lleva a cabo la gestión y aplicación de contextos de seguridad. Las clases, actividades y el servicio que intervienen en este módulo son las de la Figura 34.

Las clases *Contexto* y *ListadoContextos*, determinan los datos que componen un contexto y el manejo de una lista de contextos. Otras de las clases son *BBDD* y *BBDDContextHandler*, a través de las que, siguiendo la estructura del módulo de vulnerabilidades, se crea y maneja la tabla de contextos de la BBDD de la aplicación.

Este módulo divide su funcionalidad entre las actividades *NuevoContextActivity*, *ModificarContextActivity* y *ContextActivity* y el servicio *AplicacionContextActivity*, el cual se encarga de que la aplicación de contextos siga ejecutándose en segundo plano al salir el usuario de la misma.

Es posible separar la implementación de la funcionalidad del módulo entre la parte encargada de obtener la información de contexto y gestionar los mismos y la parte para

la detección de contextos y la aplicación de su configuración de seguridad. Estas tareas son explicadas en los siguientes apartados.

4.3.3.1. Lógica para la gestión de contextos

Para el uso de contextos se requiere de una gestión de los mismos. La aplicación permite al usuario la creación de contextos, los cuales se almacenan en la tabla de contextos de la BBDD, y sobre los que se pueden realizar modificaciones o ser eliminados. Para ello se dispone de las actividades *NuevoContextActivity*, *ModificarContextActivity*.

Un contexto se compone de su nombre o identificador, el cual es único; una localización (latitud y longitud), que define el centro del contexto; un radio, el cual permite determinar si el dispositivo se encuentra dentro del contexto; una hora de inicio y otra hora de fin, para fijar la duración del contexto; y de la clase de seguridad asociada al mismo, siendo de dos tipos, "Alta" para la habilitación del bloqueo de la pantalla y "Baja" para su no habilitación.

Los contextos se crean en el sistema a partir de los datos introducidos por el usuario a través de la interfaz *activity_nuevo_context* y obtenidos mediante el método *obtenerContexto()* de la actividad *NuevoContextActivity*. En el caso de la adquisición de los datos de localización, éstos se obtienen mediante el método *localizacionActual(View view)*, el cual utiliza los métodos *getLongitude()* y *getLatitude()* de la clase *Location* de la API de Android que proporcionan las coordenadas actuales en grados de longitud y latitud respectivamente.

Tras su creación, los contextos son almacenados en la tabla de contextos de la BBDD mediante el método *alta(NuevoContextActivity m, Contexto c)*, después de comprobarse con el método *consulta(NuevoContextActivity m, String id)* que no hay ningún otro contexto con el mismo identificador en el sistema. Ambos métodos pertenecen al manejador *BBDDContextosHandler*.

La modificación de un contexto se realiza de forma parecida a su creación. Los nuevos datos del contexto se adquieren a través del método *obtenerContexto()*, en este caso de la actividad *ModificarContextActivity*. Si se modifica la localización, se utiliza *localizacionActual(View view)* también de la actividad *ModificarContextActivity* para obtener los nuevos datos de localización con los que cambiar la situación anterior del

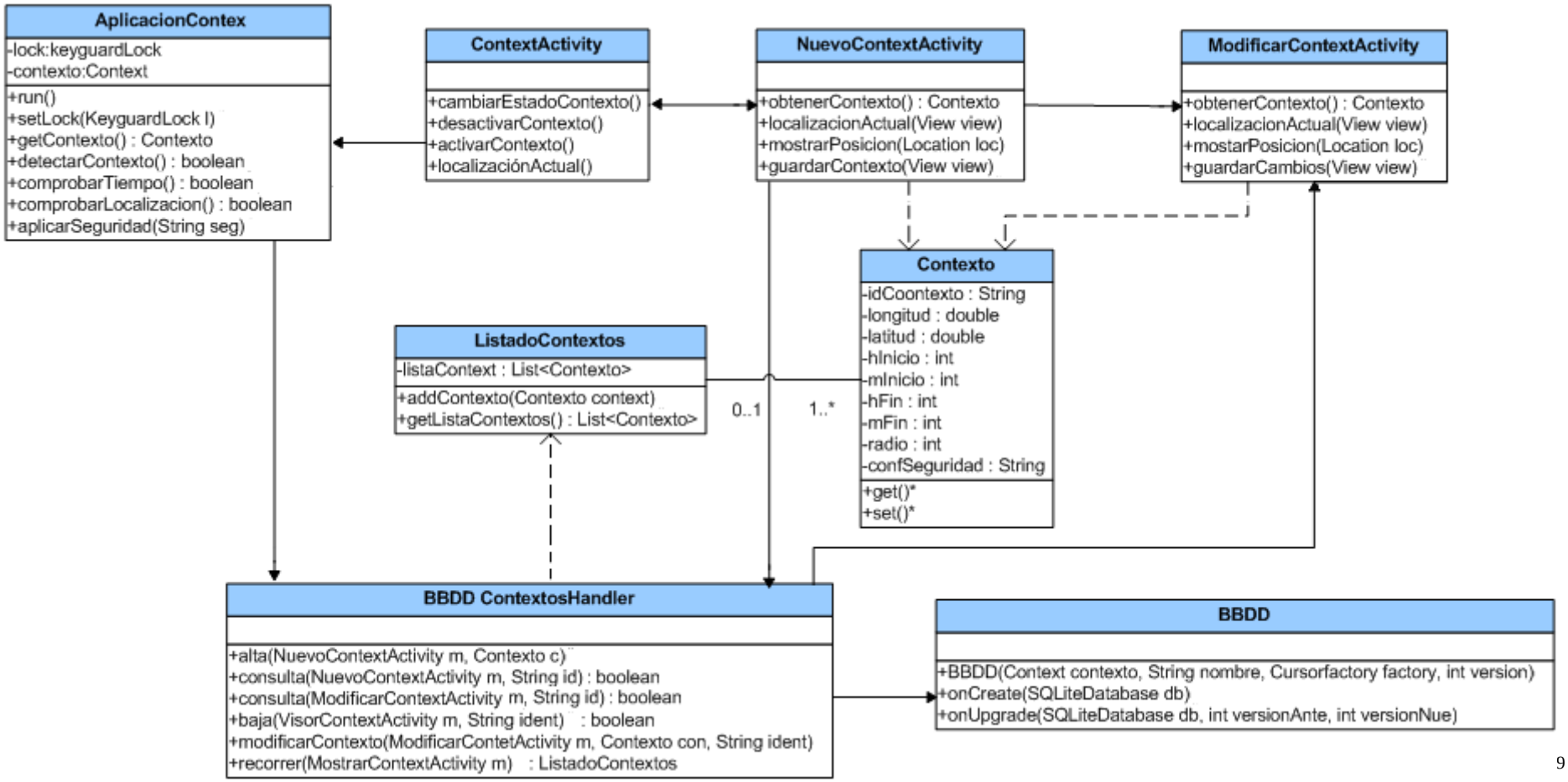

Figura 34. Clases, actividades y servicios del módulo de seguridad por contexto

[9] En la clase Contexto los métodos *get()* y *set()* se refieren a los métodos get y set correspondientes a cada uno de sus atributos

contexto a modificar. Después de esto, se almacenan los cambios realizados en la entrada correspondiente de la tabla de contextos de la BBDD para lo que se utiliza el método *modificarContexto(ModificarContextActivity m, Contexto con, String ident)* de *BBDDContextosHandler*.

Por último, para la eliminación de un contexto se emplea directamente el método *baja(VisorVulnActivity, String ident)* del manejador de la tabla de vulnerabilidades.

4.3.3.2. Lógica para la detección y aplicación de contextos

En este apartado se explica cómo está implementada la detección y aplicación de contextos de seguridad. Las acciones llevadas a cabo para la detección y aplicación de contextos se desarrollan en la actividad *ContextActivity* y el servicio *AplicaciónContext*.

Detección de un contexto

La detección de contextos consiste en comprobar si la hora y localización actuales del dispositivo se encuentran dentro de los rangos definidos en alguno de los contextos almacenados en el sistema. El flujo seguido para la implementación de la detección de contextos es el de la Figura 35.

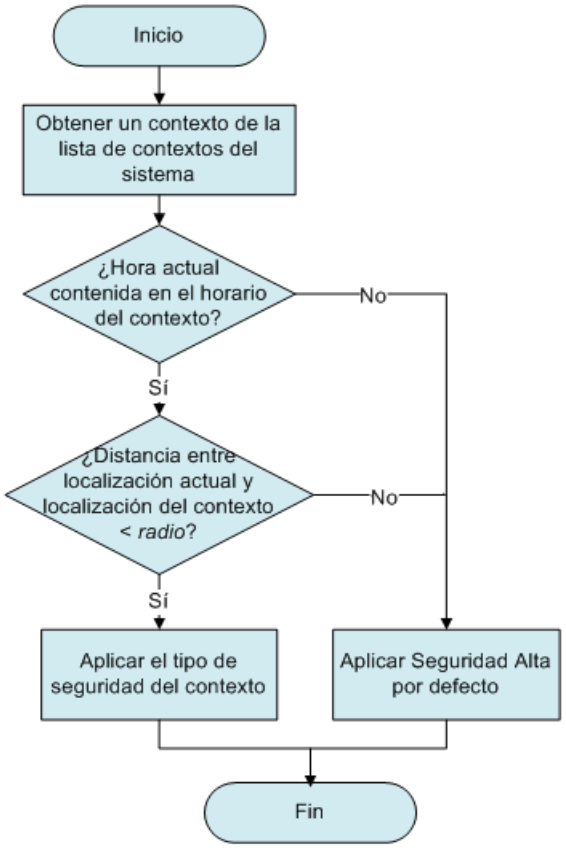

Figura 35. Diagrama de flujo para la detección de un contexto

Las detección de contextos se lleva a cabo en segundo plano. Con este fin, se desarrolla el servicio *AplicacionContext*, que ejecuta las comprobaciones necesarias dentro de su método *run()*. Este método está controlado desde la actividad *ContextActivity* a través de los métodos *activarContexto()* y *desactivarContexto()*.

Dentro de *run()*se ejecuta el método *detectarContexto()* encargado de obtiener la lista de contextos del sistema mediante el *recorrer(AplicaciónContext m)* del manejador *BBDDContextosHandler* y de decidir si el contexto analizado se corresponde con el actual. Para determinar si se ha detectado un contexto o no, *detectarContexto()* emplea los métodos *comprobarTiempo()* encargado de determinar si la hora actual se encuentra dentro del horario de un contexto y *comprobarLocalizacion()*, que calcula la distancia entre la localización actual y la del contexto estudiado para determinar si ésta es menor que su radio. Ambos métodos pertenecen también al servicio *AplicacionContext*.

Los datos relacionados con la hora y localización actuales se obtienen del dispositivo. Para conocer la hora actual del terminal se usa el método *getInstance()* del objeto *Calendar* de Android y la adquisición de los datos de localización actuales se realiza utilizando los mismos métodos que cuando se crea un nuevo contexto.

Finalmente, si se ha detectado un contexto conocido, se aplica su tipo de seguridad y en caso de no haberse detectado ninguno se aplica el tipo de seguridad "Alta" por defecto.

Las comprobaciones para la detección de contexto se llevan a cabo periódicamente cada 5 minutos, excepto si el dispositivo se encuentra en un contexto conocido y se produce un cambio en su localización, en cuyo caso se comprueba en ese mismo momento si se continúa dentro del contexto o fuera de él y se aplica la seguridad que corresponda.

Aplicación del tipo seguridad de un contexto

La aplicación de la seguridad de un contexto se encarga de modificar el bloqueo de la pantalla del terminal en función del tipo de seguridad del contexto a aplicar. El diagrama de flujo para la aplicación de la configuración de seguridad correspondiente es el que se muestra seguidamente.

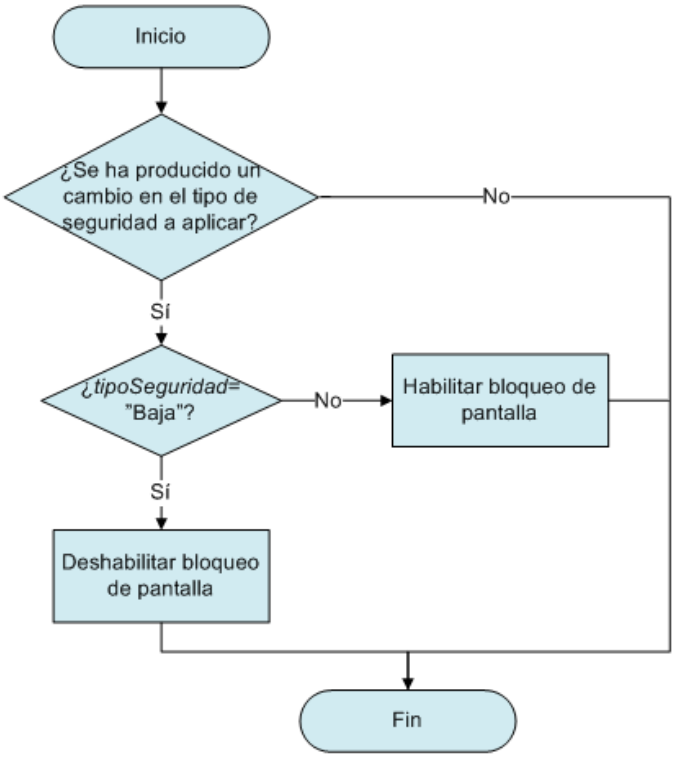

Figura 36. Diagrama de flujo para la aplicación del tipo de seguridad

A la hora de aplicar un tipo de seguridad, se comprueba si se produce un cambio con respecto al tipo de seguridad anterior y en caso de producirse, se obtiene el tipo de seguridad a aplicar en función de la que se habilita o no el bloque de la pantalla.

Este proceso se lleva a cabo a través del método *aplicarSeguridad(String seg)* del servicio *AplicacionContext*. Concretamente, para la habilitación y no habilitación del bloqueo de la pantalla del dispositivo se utilizan los métodos *reenableKeyguard()* y *disableKeyguard()* respectivamente, de un objeto de la clase *KeyguardLock*.

Capítulo 5. Pruebas

En este capítulo se comprueba el funcionamiento de la aplicación desarrollada a través de una serie de pruebas y se muestran los resultados obtenidos.

Para ello, primero se presenta el escenario utilizado para probar la aplicación y después se distribuyen las pruebas en diferentes apartados en función del módulo cuya funcionalidad se desea probar.

5.1. Entorno de pruebas

Las pruebas de la aplicación son realizadas, primeramente, en un entorno virtual mediante su simulación en varios emuladores AVD (Android Virtual Device), explicados en el anexo B.2.3, para posteriormente, llevarse a cabo en un dispositivo móvil Android real.

La parte de simulación en los AVD permite hacer pruebas en las diferentes versiones del sistema operativo y en una mayor variedad de dispositivos. Las pruebas de la aplicación se realizan para las versiones 2.2 y 4.2, excluyendo las 3.x por tratarse de versiones para *tablets*.

A la hora de realizar las pruebas del módulo de seguridad por contexto se ha tenido en cuenta que el emulador Android siempre cuenta con los mismos datos de localización al no tratarse de un dispositivo real. Para solucionar este inconveniente existen dos opciones que permiten simular la actualización de la posición del dispositivo, el envío manual de coordenadas y la utilización de listas de coordenadas [26].

El envío manual de nuevas posiciones al emulador permite simular cambios de la localización del dispositivo desde la perspectiva de DDMS, a través del menú *Emulator Control* y la pestaña *Manual* de *Location Controls*. Sin embargo, no se trata de una alternativa práctica en el caso de necesitarse hacer numerosas comprobaciones.

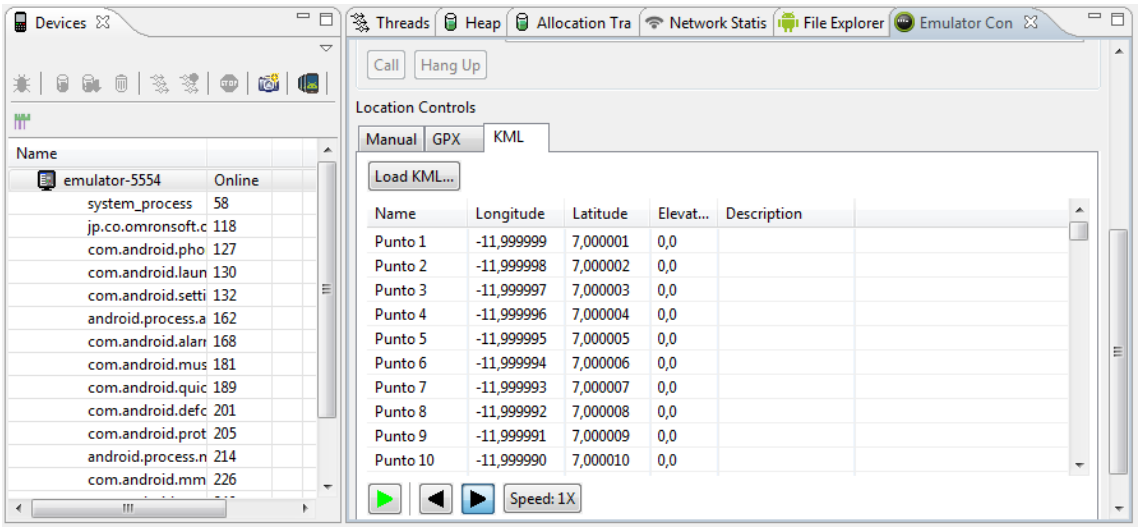

Figura 37. Perspectiva DDMS con los diferentes controles de localización

La otra opción, escogida para la realización de las pruebas, es proporcionar una lista de coordenadas enviadas al emulador una detrás de otra de forma periódica, simulando un

dispositivo en constante movimiento. La lista de coordenadas se genera en un fichero en formato GPX (GPS eXchange Format) o KML (Keyhole Markup Language) que se proporciona al emulador nuevamente desde *Location Controls* a través de la pestaña *GPX* o *KML*, en función de la extensión del fichero empleado. Una vez se ha cargado el fichero, se dispone de cuatro funcionalidades para moverse por la lista de coordenadas del mismo y determinar la velocidad de avance automático.

Finalmente, las pruebas definitivas se llevan a cabo en el dispositivo real, utilizándose para ello un Samsung Galaxy ACE.

5.2. Tablas de pruebas

Las diferentes pruebas realizadas quedan agrupadas en función del módulo del sistema al que pertenecen. Estas comprobaciones se presentan en tablas con el identificador de la prueba, su nombre, una descripción de la misma y su dependencia con los casos de uso del sistema. Además, se proporciona la información sobre su resultado.

Los códigos de identificación de las pruebas son los mostrados en la Tabla 1.

Código	Tipos de pruebas
PI-nº	Prueba de interfaz
PGV-nº	Prueba de gestión de vulnerabilidades
PSW-nº	Prueba de servidor Web
PSC-nº	Prueba de seguridad por contexto

Tabla 54. Códigos de identificación de pruebas

5.2.1. Pruebas de la interfaz de usuario

Durante el proceso de pruebas de la interfaz de usuario, se pretende comprobar que las distintas vistas y sus elementos cumplen con su función adecuadamente, es decir, que los cambios entre vistas son los que corresponden y que los datos son comprobados y mostrados en el sistema correctamente. Asimismo, se examina el funcionamiento de otros eventos como la recepción de notificaciones o los diálogos de selección.

Las pruebas de la interfaz de usuario llevadas a cabo son las presentadas en las siguientes tablas.

PI-01	Menús de selección
Descripción	Se capturan y ejecutan los eventos de selección
Dependencia	
Resultado	Correcto

Tabla 55. PI-01 Menús de selección

PI-02	Cambios entre vistas
Descripción	Las diferentes actividades son lanzadas siguiendo el orden establecido
Dependencia	
Resultado	Correcto

Tabla 56. PI-02 Cambios entre vistas

PI-03	Campos editables
Descripción	Recogen la información proporcionada por el usuario. Realizan las comprobaciones correspondientes sobre ella
Dependencia	
Resultado	Correcto

Tabla 57. PI-03 Campos editables

PI-04	Carga de datos en vistas
Descripción	La información resultante es cargada en los campos correspondientes
Dependencia	
Resultado	Correcto

Tabla 58. PI-04 Carga de campos en vistas

PI-05	Hiperenlaces
Descripción	Permiten el acceso al sitio Web indicado
Dependencia	CU-01
Resultado	Correcto

Tabla 59. PI-05 Hiperenlaces

PI-06	Recepción de notificaciones
Descripción	Se reciben notificaciones en la barra de estado a través de las que se accede a la actividad correspondiente. Las notificaciones son eliminadas al ser consultadas
Dependencia	CU-01
Resultado	Correcto

Tabla 60. PI-06 Recepción de notificaciones

PI-07	Mensajes de aviso
Descripción	Informan al producirse ciertos eventos. Desaparecen transcurridos 3.5 segundos
Dependencia	
Resultado	Correcto

Tabla 61. PI-07 Mensajes de aviso

PI-08	Diálogos de selección
Descripción	Permite la elección entre dos opciones dadas
Dependencia	
Resultado	Correcto

Tabla 62. PI-08 Diálogos de selección

5.2.2. Pruebas del módulo de gestión de vulnerabilidades

En las pruebas del módulo de gestión de vulnerabilidades se llevan a cabo las comprobaciones relativas al almacenamiento, consulta y control sobre las vulnerabilidades revisadas por el usuario, así como a los cambios en las vulnerabilidades Android publicadas a notificar al usuario.

Las primeras pruebas del módulo se realizan prescindiendo del servidor Web, es decir, con la respuesta del servidor introducida directamente en la aplicación móvil, aunque cabe señalar que estas pruebas se repitieron también una vez una vez se incorporó el servidor. Tras comprobar que el almacenamiento, resumen y búsqueda de vulnerabilidades se corresponde con el funcionamiento deseado, se introduce el servidor y se realizan las pruebas relacionadas con la petición periódica de vulnerabilidades al mismo y la recepción de respuestas con el fin de comprobar su funcionamiento en segundo plano y la notificación de cambios, debidos a la actualización o aparición de nuevas vulnerabilidades.

Las pruebas más relevantes realizadas durante y al final de la implementación de este módulo quedan recogidas en las siguientes tablas.

PGV-01	Resumen de vulnerabilidades y ver pendientes de revisión
Descripción	Se muestra gráficamente el porcentaje de vulnerabilidades revisadas, así como el número de las pendientes de ser revisadas, las cuales se pueden consultar. Los datos se actualizan con cada cambio producido
Dependencia	CU-01
Resultado	Correcto

Tabla 63. PGV-01 Resumen y ver vulnerabilidades pendientes de revisión

PGV-02	Búsqueda por fecha
Descripción	Consulta de las vulnerabilidades posteriores a la fecha especificada (incluida)
Dependencia	CU-01
Resultado	Correcto

Tabla 64. PGV-02 Búsqueda por fecha

PGV-03	Búsqueda por palabra clave
Descripción	Consulta de la palabra en el sumario de las vulnerabilidades almacenadas en la BBDD. Se pueden utilizar varias palabras clave en una misma búsqueda
Dependencia	CU-01
Resultado	Correcto

Tabla 65. PGV-03 Búsqueda por palabra clave

PGV-04	Búsqueda por fecha y palabra clave
Descripción	Filtrado de las vulnerabilidades teniendo en cuenta ambos criterios
Dependencia	CU-01
Resultado	Correcto

Tabla 66. PGV-04 Búsqueda por fecha y palabra clave

PGV-05	Histórico de vulnerabilidades
Descripción	Consulta de todas las vulnerabilidades almacenadas en la BBDD. Modificación del campo que indica si se trata de una vulnerabilidad revisada
Dependencia	CU-01
Resultado	Correcto

Tabla 67. PGV-05 Histórico de vulnerabilidades

PGV-06	Modificación del estado de una vulnerabilidad
Descripción	Cambio del estado de una vulnerabilidad de no revisada a revisada y viceversa
Dependencia	CU-01
Resultado	Correcto

Tabla 68. PGV-06 Modificación del estado de una vulnerabilidad

PGV-07	Activación/ Desactivación de la recepción de vulnerabilidades
Descripción	Activación y desactivación de la recepción de vulnerabilidades en el sistema
Dependencia	CU-01
Resultado	Correcto

Tabla 69. PGV-07 Activación/ Desactivación recepción de vulnerabilidades

PGV-08	Acceso a la solución de una vulnerabilidad
Descripción	La información de una vulnerabilidad contiene un hiperenlace por el que se accede a la solución dada por el fabricante del producto con la vulnerabilidad
Dependencia	CU-01
Resultado	Correcto

Tabla 70. PGV-08 Acceso solución

PGV-09	Obtención y notificación de nuevas vulnerabilidades
Descripción	Se notifica si se han recibido nueva información de vulnerabilidades, indicando el número de vulnerabilidades recibidas
Dependencia	
Resultado	Correcto

Tabla 71. PGV-09 Obtención y notificación de vulnerabilidades

PGV-10	Funcionamiento en segundo plano
Descripción	La actualización de la información sobre vulnerabilidades sigue funcionando al salir el usuario de la aplicación
Dependencia	
Resultado	Correcto

Tabla 72. PGV-10 Funcionamiento en segundo plano

5.2.3. Pruebas del servidor Web

Para probar el servidor Web, en primer lugar, se comprueba si la comunicación del servidor con el dispositivo móvil y la NVD se establece correctamente. Para ello, se deja corriendo el servidor Web a la espera de peticiones, primero desde un navegador y

después directamente desde el terminal móvil. Asimismo se comprueba que el parámetro con la versión Android en las peticiones al servidor es extraído correctamente en el mismo. Después, se prueba la conexión con la NVD, descargándose el fichero de vulnerabilidades en el servidor.

Tras ello se verifica si la generación de la respuesta en el servidor a partir del fichero recibido de la NVD es correcta, para lo que se comprueban los pasos para el procesado del fichero y la construcción de la lista de vulnerabilidades a enviar.

Estas pruebas de comprobación del servidor Web se presentan en las tablas mostradas a continuación.

PSW-01	**Conexión cliente-servidor web**
Descripción	Se establece conexión HTTP entre la aplicación y el servidor Web correctamente. El envío incluye parámetros en la petición
Dependencia	
Resultado	Correcto

Tabla 73. PSW-01 Conexión cliente-servidor Web

PSW-02	**Conexión servidor Web- servidor NVD**
Descripción	Se establece conexión HTTP entre el servidor y la NVD correctamente. Se descarga del fichero de vulnerabilidades de la NVD
Dependencia	
Resultado	Correcto

Tabla 74. PSW-02 Conexión servidor Web-servidor NVD

PSW-03	**Cierre de las conexiones**
Descripción	Se cierran las conexiones establecidas
Dependencia	
Resultado	Correcto

Tabla 75. PSW-03 Cierre conexiones

PSW-04	**Identificación de los campos**
Descripción	Se procesa el fichero recibido mediante la identificación de los campos de las vulnerabilidades
Dependencia	
Resultado	Correcto

Tabla 76. PSW-04 Identificación campos

PSW-05	Generación de la lista de vulnerabilidades
Descripción	Se extraen solamente los campos relevantes de las vulnerabilidades con la versión Android indicada anteriormente en la petición al servidor y se crea una lista de vulnerabilidades como respuesta
Dependencia	
Resultado	Correcto

Tabla 77. PSW-05 Generación lista vulnerabilidades

5.2.4. Pruebas del módulo de seguridad por contexto

Con la realización de las pruebas del módulo de seguridad por contexto, se comprueba el funcionamiento de los contextos de seguridad, tanto en lo relativo a su administración como a la aplicación del tipo de seguridad correspondiente a los mismos.

Para ello, primero se prueba la correcta interacción con la BBDD mediante la creación y consulta de contextos para su modificación o eliminación. Una vez el manejo de los contextos es correcto, se pueden realizar las comprobaciones relacionadas con la identificación del contexto actual y por último, la verificación de la aplicación de las medidas de seguridad que le correspondan.

Debido a la necesidad de simular cambios de contexto, y por tanto de posición geográfica, se utiliza un fichero KML con 1000 coordenadas aleatorias, encontrándose varias de ellas dentro de contextos conocidos dentro de la aplicación con el fin de comprobar su correcta detección.

Las pruebas realizadas en este módulo son las de las tablas siguientes.

PSC-01	Creación de un nuevo contexto
Descripción	Se crea un nuevo contexto con los datos especificados
Dependencia	CU-02
Resultado	Correcto

Tabla 78. PSC-01 Creación contexto

PSC-02	Consulta de los datos de un contexto
Descripción	Obtención de la lista con los identificadores de los contextos previamente definidos en la aplicación y visualización de todos los datos de un contexto
Dependencia	CU-02
Resultado	Correcto

Tabla 79. PSC-02 Consulta datos contexto

PSC-03	Modificación de un contexto
Descripción	Posibilidad de modificación de todos o algunos de los datos de un contexto
Dependencia	CU-02
Resultado	Correcto

Tabla 80. PSC-03 Modificación contexto

PSC-04	Eliminación de un contexto
Descripción	Se elimina el contexto indicado
Dependencia	CU-02
Resultado	Correcto

Tabla 81. PSC-04 Eliminación contexto

PSC-05	Activación/ Desactivación de la monitorización de contextos
Descripción	Activación y desactivación de la utilización de contextos de seguridad
Dependencia	CU-02
Resultado	Correcto

Tabla 82. PSC-05 Activación/Desactivación del uso de contextos

PSC-06	Obtención del contexto actual y detección de contextos
Descripción	Se obtiene la situación y la hora actuales del dispositivo y se comprueba si el contexto actual se corresponde con un contexto previamente definido
Dependencia	
Resultado	Correcto

Tabla 83. PSC-06 Contexto actual y detección de contextos

PSC-07	Configuración de seguridad según contexto
Descripción	Se aplica el tipo de seguridad en función del contexto en el que se sitúa el terminal
Dependencia	
Resultado	Correcto

Tabla 84. PSC-07 Aplicación de seguridad según contexto

PSC-08	Funcionamiento en segundo plano
Descripción	La detección de contextos sigue funcionando al salir el usuario de la aplicación
Dependencia	
Resultado	Correcto

Tabla 85. PSC-08 Funcionamiento en segundo plano

5.3. Matriz RSF x (PGV+PSC)

Para terminar, se presenta una matriz que relaciona los requisitos funcionales con las pruebas del módulo de gestión de vulnerabilidades y las pruebas del módulo de seguridad por contexto. A través de esta matriz, mostrada en la Tabla 86, se asegura que todos los requisitos funcionales quedan examinados en las pruebas llevadas a cabo.

Dado que el servidor Web no se trata de un requisito inicial de la aplicación, si no de un elemento incorporado durante la implementación del sistema, las comprobaciones que verifican su funcionamiento no dependen directamente de ningún requisito funcional.

	PGV-01	PGV-02	PGV-03	PGV-04	PGV-05	PGV-06	PGV-07	PGV-08	PGV-09	PGV-10	PSC-01	PSC-02	PSC-03	PSC-04	PSC-05	PSC-06	PSC-07	PSC-08
RSF-01							X											
RSF-02									X									
RSF-03									X									
RSF-04		X																
RSF-05			X															
RSF-06				X														
RSF-07					X													
RSF-08								X										
RSF-09						X												
RSF-10	X																	
RSF-11	X																	
RSF-12															X			
RSF-13											X							
RSF-14												X						
RSF-15													X					
RSF-16														X				
RSF-17																X		
RSF-18																	X	
RSF-19										X								X

Tabla 86. Matriz RSF x (PGV + PSC)

Capítulo 6. Histórico del proyecto

En este capítulo se recogen las etapas llevadas a cabo para la realización de este proyecto.

Cada una de las etapas es explicada en detalle a lo largo de la sección, al final de la cual se presentará un resumen de la historia del proyecto.

6.1. Etapas del proyecto

Este proyecto se divide en varias fases, concretamente en cuatro etapas: la etapa de definición de objetivos, de planteamiento inicial, de implementación del sistema y por último la de documentación. Dentro de cada una de éstas, se realizaron diferentes tareas, se encontraron algunos problemas y finalmente se obtuvieron unos resultados. Asimismo, aunque no se contemplen en este apartado por haber sido explicadas en el capítulo 5, se fueron realizando pruebas en cada fase. A continuación se explica cada una de las etapas del proyecto.

6.1.1. Etapa I: Definición de objetivos

- **Tareas realizadas**

  La idea de partida para la realización de este proyecto fue la de contribuir a un mayor control y adaptabilidad de la seguridad en terminales móviles. Concretamente se planteó el desarrollo de un complemento de seguridad para dispositivos con sistema operativo Android. Con este propósito se definieron varios objetivos, como la aplicación de contextos de seguridad y la utilización de información sobre vulnerabilidades existentes en aplicaciones Android.

  Una vez establecidos los objetivos, se inició una fase de documentación para conocer las posibilidades para la consecución de éstos y el entorno de trabajo a utilizar.

  Para la obtención de la información sobre vulnerabilidades, se planteó, por un lado, la incorporación y utilización de un NIDS, y por otro, su extracción de la NVD del NIST. Para valorar el uso de un NIDS, se buscaron y consultaron varios documentos con la descripción de NIDSs para Android, concretamente los recogidos en [19], [20], [21] y [22], y se estudió la disponibilidad y complejidad de cada uno de ellos. Asimismo, se analizaron las fuentes de información de la NVD [25] y las posibilidades para la descarga de sus datos.

  En cuanto a los contextos de seguridad, se examinaron diversos documentos, siendo los principales [7] y [27], con el objetivo de extraer ideas y explorar diferentes vías antes de su implementación. Asimismo, se utilizó la página de desarrolladores [3] para obtener un primer acercamiento sobre las aportaciones de las librerías de Android a la seguridad por contexto.

Con el fin de conocer las herramientas para la programación en Android, se instaló el IDE Eclipse junto con el SDK de Android, ambos explicados en B.2.2 y B.2.3 respectivamente. Además, se consultó algún libro sobre la estructura e implementación en Android [5]. Asimismo, como inicio a la programación de aplicaciones en Android, se analizaron y ejecutaron varios de los ejemplos disponibles en [3].

- **Problemas encontrados**

  Durante esta etapa, se planteó como problema la no existencia de aplicaciones NIDS con un desarrollo definitivo y disponibles para su uso directamente en Android. Esto implicó que su utilización no se tratase de algo sencillo, ya que requería de modificaciones en su código y de una compleja instalación en función del *hardware* en el que se fuera a utilizar, quedando fuera de las líneas de trabajo de este proyecto.

- **Resultados obtenidos**

  Como resultado de esta primera fase, se descartó la utilización de un NIDS como fuente de información debido a su no disponibilidad, resultando los datos almacenados en la NVD como la mejor opción para obtención de la información sobre vulnerabilidades de seguridad.

  Además, se confirmó la posibilidad de la utilización de contextos de seguridad mediante el empleo de la información proporcionada por el dispositivo a través del uso de las librerías indicadas para ello.

6.1.2. Fase II: Planteamiento inicial

- **Tareas realizadas**

  El problema encontrado en la etapa anterior, estableció como definitivo el uso de la NVD como fuente para la obtención de la información sobre vulnerabilidades, por tratarse de una opción completa y fiable para la extracción de los datos sobre vulnerabilidades a falta de NIDS sencillos que utilizar.

  La información sobre cada vulnerabilidad en la NVD es extensa lo que llevó al análisis de la misma, para determinar cuáles eran los parámetros más relevantes que permitieran informar de forma clara al usuario sobre las anomalías en las aplicaciones disponibles para su sistema operativo. Este análisis permitió establecer

la estructura de una vulnerabilidad dentro del sistema implementado y las acciones para la gestión de las mismas por parte del usuario.

También, se concretaron los parámetros finales que se utilizarían para la definición de contextos y se analizó la información de entorno proporcionada por terminal para su utilización en la detección de contextos. Asimismo, se estudiaron y establecieron las posibilidades para los cambios en la configuración de seguridad del dispositivo.

Estas decisiones permitieron determinar un esquema inicial con las funciones que sería necesario llevar a cabo para la detección y el control de los contextos de seguridad.

- **Problemas encontrados**

  A pesar de la facilidad para la obtención de la información de la NVD a través descarga de un fichero mediante una petición HTTP, este fichero recoge las vulnerabilidades presentes en todo tipo de sistemas operativos sin que éstas estén clasificadas. Este hecho dificultó la descarga de únicamente las vulnerabilidades relacionadas con Android.

  Además de para la extracción de solamente las vulnerabilidades referentes a aplicaciones para Android, tal y como se observó, la cantidad de datos sobre cada vulnerabilidad en el fichero de la NVD es grande, haciéndose necesario un procesado de la misma. En este sentido se planteó la necesidad de un nuevo elemento intermedio entre el terminal móvil y la BBDD del NIST, encargado de procesar la información antes de su envío a la aplicación cliente.

  Respecto a los parámetros de seguridad, las posibilidades que ofrece la API de Android para la adaptabilidad de la seguridad en función del contexto son limitadas. Algunos de los parámetros configurables están relacionados con el uso de cifrado de los datos o la política de *passwords*, los cuales no deberían de ser modificados cada vez que se produzca un cambio de contexto.

  Por otro lado, en lo referente al bloqueo de la pantalla del dispositivo, es posible retrasar el tiempo de bloqueo del terminal, sin embargo, no existe la posibilidad de cambio automático entre diferentes métodos de bloqueos de la pantalla del dispositivo sin que sea necesaria la interacción del usuario, es decir, a modo de ejemplo, no es posible el paso de un bloqueo de deslizamiento a un bloqueo numérico y viceversa, lo que limita aún más la configuración de seguridad automática en función del contexto.

- **Resultados obtenidos**

  Con la idea de utilizar un nuevo elemento intermedio de procesado, se terminó de modelar el sistema. El elemento escogido fue un servidor Web, de manera que la aplicación se componía finalmente de tres módulos funcionales junto con el módulo de interfaz: el módulo de seguridad por contexto, el módulo de vulnerabilidades de seguridad y el módulo del servidor Web.

  El módulo de seguridad por contexto se encargaría de la gestión y aplicación de contextos de seguridad mientras que el módulo de vulnerabilidades de seguridad junto con el servidor Web proporcionaría la información sobre vulnerabilidades y su gestión.

  En cuanto a los parámetros de entorno para la detección de un contexto, se determinó el uso de la información de localización y hora del dispositivo y respecto a los cambios en la configuración de seguridad, frente a la no posibilidad de cambio entre diferentes métodos de bloqueo se optó por la habilitación y no habilitación del uso del bloqueo de la pantalla del terminal.

  Además, en el diseño del sistema, se contó con que la actualización de vulnerabilidades y la aplicación de contextos se realizaría de forma transparente al usuario, es decir, se trata de procesos corriendo en segundo plano.

6.1.3. Fase III: Implementación del sistema

6.1.3.1. Implementación del bloque de gestión de vulnerabilidades

- **Tareas realizadas**

  En primer lugar se desarrolló la parte de comunicación con el servidor Web para la adquisición de la información de vulnerabilidades. Como servidor, se optó por utilizar Apache Tomcat debido a su sencillez e integración en plataformas con JVM. Se llevó a cabo su instalación y se procedió a la documentación sobre servlets [10]. Tras ello, se programó el servlet para la descarga del fichero de la NVD con la información sobre vulnerabilidades *software*, dirigida a partir de las peticiones lanzadas, en segundo plano, desde el cliente Android. Este protocolo de comunicación entre el dispositivo Android y el servidor Web implementado también fue definido e implementado en esta fase.

  Dentro de la implementación del servidor Web, también se realizó el procesado de la información del fichero descargado para la extracción, en exclusiva, de las

vulnerabilidades de la versión del sistema operativo Android del cliente y para la obtención de los campos de información más relevantes de cada vulnerabilidad, siguiendo la estructura fijada durante la fase II. A este fin se generó y utilizó un *parser XML*.

Una vez con la información sobre vulnerabilidades en el dispositivo móvil, se implementaron las tareas para la comprobación de si las vulnerabilidades recibidas se referían a aplicaciones que se encontraran instaladas en el terminal y finalmente su almacenamiento en la tabla de vulnerabilidades de la BBDD de la aplicación.

Para terminar, se desarrollaron los métodos necesarios para el control y consulta de la información de las vulnerabilidades por parte del usuario.

- **Problemas encontrados**

  En la NVD, la descripción de cada una de las vulnerabilidades presentada en una aplicación se realiza en el sumario de la misma. Este hecho dificulta la resolución de éstas en el terminal ya que se hace necesario el análisis semántico del sumario para ello. Este problema limitó la gestión de vulnerabilidades, eliminando la posibilidad de actuación de este bloque, de forma que el programa sólo puede gestionarlas a través de las acciones realizadas por el usuario. Es decir, la responsabilidad para la revisión y eliminación de estas vulnerabilidades se delega en el usuario, a través de un hiperenlace a la solución propuesta por el fabricante.

- **Resultados obtenidos**

  Tras la finalización de la implementación del módulo, se obtiene un gestor de vulnerabilidades que dispone de información actualizada procedente de la NVD del NIST y que permite que el usuario mantenga un control sobre las posibles vulnerabilidades en las aplicaciones del sistema a través de una interfaz sencilla para su visualización y consulta.

  Las vulnerabilidades mostradas disponen de un enlace con la solución propuesta para que las que se encuentren en el terminal puedan ser solucionadas y la posibilidad de dejar constancia de ello, lo que permite presentar un resumen de seguridad con el cálculo del porcentaje de vulnerabilidades que hayan sido revisadas.

6.1.3.2. Implementación del bloque de seguridad por contexto

- **Tareas realizadas**

  Primeramente, se implementó la obtención de la información necesaria procedente del dispositivo móvil, es decir, su localización y hora actual para la definición y detección de contextos.

  Tras disponer de dicha información, se desarrollaron los métodos para la creación y gestión de contextos en su tabla correspondiente de la BBDD de la aplicación, y se implementaron las tareas que, ejecutadas en segundo plano, se encargan de las comprobaciones para la identificación de los sucesivos contextos en los que se puede encontrar el terminal.

  Por último, se crearon los métodos para la aplicación de la configuración de seguridad del contexto en el que está dispositivo, es decir, de los dos tipos de seguridad del sistema relacionados con la habilitación o no del uso del bloqueo de la pantalla del terminal. Primero se realizaron pruebas fuera del módulo y después se procedió a su integración en el mismo.

  Problemas encontrados

  Aunque la primera opción en la implementación de la detección de contextos era la utilización de *Geofences* [28], por tratarse de objetos definidos en Android para el establecimiento de áreas de interés, finalmente, se optó por usar la función *distanceBetween.* Esta función, que proporciona el cálculo directo de la distancia entre dos coordenadas, sustituyó el uso de *Geofences* debido a su simplicidad y facilidad de integración en el sistema de gestión de contextos ya implementado y su compatibilidad con todas las versiones de Android, a pesar de requerir algo de lógica adicional a desarrollar.

- **Resultados obtenidos**

  Con la realización de este módulo, se integran los datos del entorno como información para el establecimiento niveles de seguridad en el dispositivo y se obtiene la funcionalidad necesaria para la gestión y detección de contextos de seguridad.

  La seguridad de los contextos definidos por el usuario son aplicados en función de la localización del dispositivo y hora actuales, existiendo los tipos de configuración de

seguridad bajo (no habilitación del bloqueo de la pantalla) y alto (habilitación del bloqueo).

6.1.4. Etapa IV: Documentación

En la etapa de documentación se recogió en esta memoria todo el trabajo llevado a cabo para el desarrollo de este proyecto.

6.2. Resumen

Para terminar, se obtiene una visión resumida de las etapas del proyecto junto a su planificación temporal. El proyecto se llevó a cabo entre los meses de noviembre de 2012 y septiembre de 2013 a excepción de enero y junio. El resumen del mismo, se presenta en la Tabla 87.

Etapa	Descripción		Duración (meses)
Etapa I	Definición de objetivos		2
Etapa II	Planteamiento inicial		2
Etapa III	Implementación del sistema	Bloque de seguridad por contexto	1,5
		Bloque de gestión de vulnerabilidades	1,5
Etapa IV	Documentación		2

Tabla 87. Resumen de la duración de las etapas del proyecto

La cantidad de tiempo requerido en cada parte tiene relación directa con el esfuerzo necesario para su realización. Así, la parte más laboriosa se concentró en las primeras etapas en las que fue necesario investigar y plantear el desarrollo de la aplicación, aunque durante la implementación de la misma existió la necesidad de consultar más documentación lo que alargó este proceso hasta casi igualarlo al de las primeras etapas de diseño.

El reparto de tiempo entre tareas fue, por tanto, bastante parecido como puede observarse en el gráfico presentado en la Figura 38.

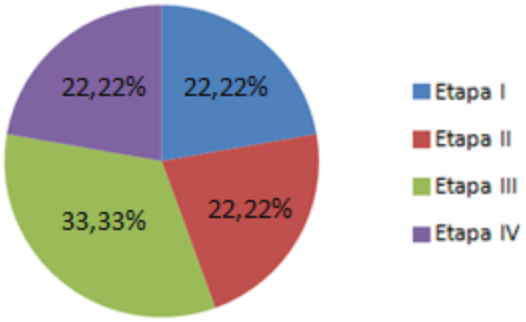

Figura 38. Duración de las etapas del proyecto en porcentaje

Capítulo 7. Conclusiones y líneas futuras

7.1. Conclusiones

Los dispositivos móviles inteligentes o *smartphones* se han convertido en elementos indispensables en el día a día. A ellos se confía datos confidenciales y personales constantemente. Asimismo, la necesidad de estar continuamente conectado a la red ha contribuido a la amplia utilización de los *smartphones* pero sin reparar en la importancia de la seguridad en los mismos, sí existente en otros dispositivos como los ordenadores personales.

Android es el sistema operativo para dispositivos móviles más utilizado en la actualidad, con una cuota de mercado en torno al 70%. Sin embargo su rápido crecimiento en tan solo unos pocos años lleva consigo la falta de mecanismos de seguridad para el control y la adaptabilidad de su seguridad.

Este proyecto contribuye a la seguridad en Android mediante el desarrollo de una aplicación que, a partir de la información proporcionada por una BBDD de vulnerabilidades y el entorno, permite el conocimiento del riesgo al que se encuentra dispuesto el dispositivo con el fin de controlar las vulnerabilidades presentes en el mismo y para tomar las medidas de seguridad correspondientes al contexto en el que se encuentre. Esto se ha logrado mediante la implementación de un gestor de vulnerabilidades y un módulo de seguridad por contexto.

El funcionamiento de la aplicación continúa en segundo plano al abandonarse la misma, con el fin de que la actualización de vulnerabilidades y la aplicación de contextos de seguridad se realice de forma eficaz y no se vea interrumpida, pero permitiendo su activación y desactivación por parte del usuario.

El sistema desarrollado tiene en cuenta las restricciones actuales de los terminales móviles. Por ello, la aplicación es ligera, de 400 Kb, y además externaliza el procesamiento de la información de vulnerabilidades a un servidor Web. La aplicación también es extensible, siendo posible la adición de nuevos módulos que proporcionen mayor información relacionada con la seguridad en el dispositivo.

Por último, la aplicación se ha logrado desarrollar en un entorno creado exclusivamente con herramientas de software libre, lo que facilita el acceso a la misma para su modificación y la introducción de mejoras de una forma sencilla y asequible. Asimismo se ha comprobado su correcta funcionalidad en varias de las versiones Android existentes.

7.2. Líneas futuras de trabajo

Para finalizar, se exponen algunas de las posibles vías futuras para la ampliación de este proyecto con el fin de introducir funcionalidades que contribuyan a obtener una aplicación de seguridad para Android más completa. Estas propuestas se presentan a continuación:

- **Creación o introducción de un NIDS**

  La utilización de un NIDS en el dispositivo facilitaría la detección de vulnerabilidades en las aplicaciones instaladas en él. Su detección sería más rápida por ser directa en comparación con la extracción de la información de la NVD del NIST y la necesidad de su procesado.

  Al tratarse Android de un sistema operativo aún reciente, no existen NIDS disponibles para él, a excepción de algunas versiones de prueba de difícil instalación. Por ello, una línea futura de trabajo es el diseño y la creación de un NIDS para Android, el cual podría ser incorporado dentro de la gestión de vulnerabilidades de la aplicación y trabajar conjuntamente con la información de la NVD para el control del riesgo en el sistema. Otra alternativa es el estudio, realización de las modificaciones oportunas e instalación de unos de los NIDS ya creados para su utilización, igualmente, en la detección de vulnerabilidades en el sistema.

- **Análisis semántico de las vulnerabilidades**

  La aplicación desarrollada obtiene los datos sobre vulnerabilidades de seguridad en Android procedente de la NVD del NIST. A pesar de que el código CWE ofrece una clasificación de la vulnerabilidad en función del tipo de debilidad *software* de que se trata, la explicación concreta de cada una de ellas se recoge en su sumario, el cual consiste en una cadena de caracteres con la descripción de la vulnerabilidad, lo que dificulta su estudio.

  Se propone, por tanto, la introducción del análisis semántico de la información proporcionada en el sumario de cada vulnerabilidad mediante la búsqueda, estudio y utilización de herramientas de análisis semántico de textos con el fin de que la aplicación sea capaz de examinar los datos pertenecientes a cada vulnerabilidad de forma detallada para que, a partir de ellos, exista la posibilidad de generar una lógica de actuación.

- **Automatización de la resolución de vulnerabilidades**

  A raíz del planteamiento sobre el análisis semántico de las vulnerabilidades, se propone la automatización de su resolución. Para ello, se deben estudiar los posibles métodos de mitigación de vulnerabilidades a implementar dentro del dispositivo. Asociando estos métodos al análisis semántico, sería posible la solución de las vulnerabilidades presentes en el dispositivo de forma transparente al usuario, es decir, sin necesidad de su intervención.

- **Servidor Web como servicio para la gestión de vulnerabilidades**

  Una posible vía de trabajo es la ampliación de la funcionalidad del servidor Web para que no sólo se encargue del procesado de la información sobre vulnerabilidades sino también de su detección en diferentes dispositivos, conformando un servicio de gestión de vulnerabilidades online que reduciría aún más la cantidad de información a tratar en el terminal móvil.

  Para ello, se plantea la creación en el servidor de una cuenta por cada terminal móvil que utilice de sus servicios, siendo necesaria la introducción de la autenticación de los mismos en las conexiones con el servidor. Cada cuenta mantendría la información sobre las aplicaciones instaladas en el dispositivo móvil correspondiente y el servidor realizaría las comprobaciones para determinar si las vulnerabilidades recibidas de la NVD se refieren a alguna de esas aplicaciones, informándose al dispositivo en cuestión de solamente ellas.

- **Incremento de la granularidad en la detección de contextos**

  En el sistema implementado, los contextos de seguridad se definen a través de una posición geográfica, un radio y una hora de inicio y fin. Sin embargo, una posible línea de mejora sería la obtención de una detección de contextos más fina con la introducción de otros parámetros. En este sentido es posible considerar la familiaridad del contexto [1] cuyo cálculo se lleva a cabo teniendo en cuenta la asiduidad de otros dispositivos en el contexto.

- **Creación asistida de contextos**

  Como paso para la simplificación de la gestión de contextos de seguridad, se plantea el uso una herramienta que proporcione datos sobre el entorno de localización del terminal con el fin de conocer automáticamente la información de contexto, la cual permita establecer el nivel de riesgo del entorno sin que sea necesario que el usuario defina los contextos de seguridad previamente o bien para complementar su información.

- **Nuevas medidas de seguridad en Android**

  La aplicación permite establecer dos tipos de seguridad, alta y baja, en relación con la existencia o no de bloqueo del teléfono. Sin embargo, es deseable introducir una mayor granularidad, es decir, un mayor número de diferentes niveles de seguridad. Por ello, una posible mejora sería continuar examinando la modificación automática de nuevos parámetros de configuración de seguridad en el sistema operativo Android, aparte del patrón de bloqueo utilizado.

Anexo A. Presupuesto

Se procede al cálculo del presupuesto de este proyecto. Los costes que han conllevado la realización de este proyecto se desglosan en dos bloques, el de coste de personal y el de coste de material e infraestructura. El conjunto de ambos costes se presentará en el coste total que constituirá el presupuesto final.

A.1. Costes de personal

Los costes de personal se reducen a un ingeniero de telecomunicación para la realización del proyecto. Para el cálculo de su dedicación, se estima una media de trabajo de 5 horas diarias durante los meses de noviembre de 2012 y septiembre de 2013 a excepción de los meses de enero y junio, suponiendo un total de 1080 horas trabajadas.

Para determinar el salario del ingeniero de telecomunicación se ha tenido en cuenta el sueldo medio de este perfil profesional en los últimos meses, estableciéndose un salario de 21,7 €/hora, por lo que el coste final en personal asciende a un total de 23.436 €.

Presupuesto en personal			
Concepto	**Dedicación (h)**	**Salario (€/h)**	**Coste (€)**
Ingeniero de Telecomunicación	**1.080**	**21,7**	23.436
TOTAL			**23.436**

Tabla A-1. Costes de personal

A.2. Costes de material e infraestructura

Para el desarrollo de este proyecto se ha precisado de material hardware y software. Los materiales hardware empleados han sido los presentados a continuación.

- Ordenador portátil SONY VAIO, con sistema operativo Windows 7, cuyo coste asciende a 630 €.

- Dispositivo Samsung GALAXY ACE, con sistema operativo Android versión 2.3, con un coste de 227 €.
- Cable microUSB, disponible junto al dispositivo móvil utilizado.

En el caso del software no se ha incurrido en gastos debido a la utilización de programas de libre distribución.

En cuanto a la infraestructura necesaria a lo largo de la realización del proyecto se ha requerido la mostrada seguidamente.

- Conexión fija a Internet, ADSL a 6MB de velocidad.
- Conexión móvil a Internet, tarifa de datos 200MB al mes.

En la Tabla A-2 puede observarse el desglose de los costes en material.

Presupuesto en material e infraestructura					
Concepto	**Unidades**	**Precio (€/unidad)**	**Dedicación (meses)**	**Período de depreciación**	**Coste imputable[10] (€)**
Ordenador portátil	1	630	9	60	94,5
Dispositivo móvil Android	1	227	1	60	3,78
Cable microUSB	1	0[11]	9	60	0
Conexión fija a Internet	1	48.8	9	-	439,2
Conexión móvil a Internet	1	15	1	-	15
TOTAL					**552,48**

Tabla A-2. Costes de material e infraestructura

[10] La fórmula utilizada para el cálculo de la amortización es $\frac{A}{B} x C x D$ donde
A = nº de meses desde la fecha de facturación en el que el equipo es utilizado
B = período de depreciación
C = coste del equipo
D = % del uso que se dedica al proyecto (habitualmente 100%)

[11] El cable microusb, como se ha indicado, se corresponde con el incluido junto al dispositivo móvil.

A.2. Coste Total

Una vez conocidos los gastos incurridos en personal y en material e infraestructura, es posible calcular el valor final del presupuesto. Este resultado se obtiene de la suma de ambos costes. El presupuesto total asciende a la suma de 23.985,82 €.

La Tabla A-3 recoge el resumen del presupuesto final del proyecto.

Presupuesto final	
Costes de personal	23.436
Costes de material e infraestructura	552,48
TOTAL	**23.988,48**

Tabla A-3. Coste total

Anexo B. Entorno de trabajo

Se procede a presentar las herramientas utilizadas para el desarrollo de este proyecto. En primer lugar se muestran las herramientas hardware. Después, se explican las de tipo software así como su configuración.

B.1. Herramientas hardware

Las herramientas hardware empleadas a lo largo de este proyecto son las siguientes:

- Un **ordenador portátil** personal.
- Un **smartphone** Samsung GALAXY ACE con sistema operativo **Android**.
- Un **cable microUSB**.

B.2. Herramientas software

Las herramientas software utilizadas son las citadas a continuación:

- La **Máquina virtual Java** (JVM).
- El entorno de desarrollo **Eclipse**.
- El **SDK Android**.
- El contenedor de Servlets **Apache Tomcat 6.0**.

En los siguientes apartados se detalla en qué consisten estos elementos y los aspectos de su configuración más relevantes.

B.2.1 Java Development Kit

Para poder desarrollar un programa en Java se requiere del JDK (Java Development Kit) que contiene el JRE que actúa como intermediario entre el sistema operativo y Java y la JVM para la ejecución de las aplicaciones Java en el equipo. Su descarga puede realizarse desde la página oficial de java [29].

B.2.2 Eclipse

Eclipse es una multiplataforma de desarrollo *software* de código abierto. Se trata de un conjunto de servicios que construyen un entorno de desarrollo con IDEs (Integrated Development Environment) y compiladores a partir de componentes conectados o *plugins*. Puede descargarse directamente desde su página oficial [30], desde la que se pueden obtener también algunos *plugins*. La versión de Eclipse utilizada durante la realización de este proyecto ha sido Eclipse Helios.

Eclipse es la herramienta más adecuada para el desarrollo de aplicaciones Android en la actualidad ya que, además de permitir la programación en el lenguaje Java, dispone de un *plugin* exclusivo para Android. Este plugin facilita la implementación de las aplicaciones y genera los archivos .apk de las mismas.

B.2.3. SDK Android

El SDK proporciona la plataforma necesaria para el desarrollo de la aplicación Android. Su última versión puede ser descargada desde la página oficial de desarrolladores de Android [3] en un fichero *.zip* que debe ser descomprimido en una ruta conocida que servirá de referencia al SDK. Entre las principales herramientas proporcionadas por el SDK de Android se encuentran un emulador y soporte para la detección de errores. Además dispone de ejemplos y documentación.

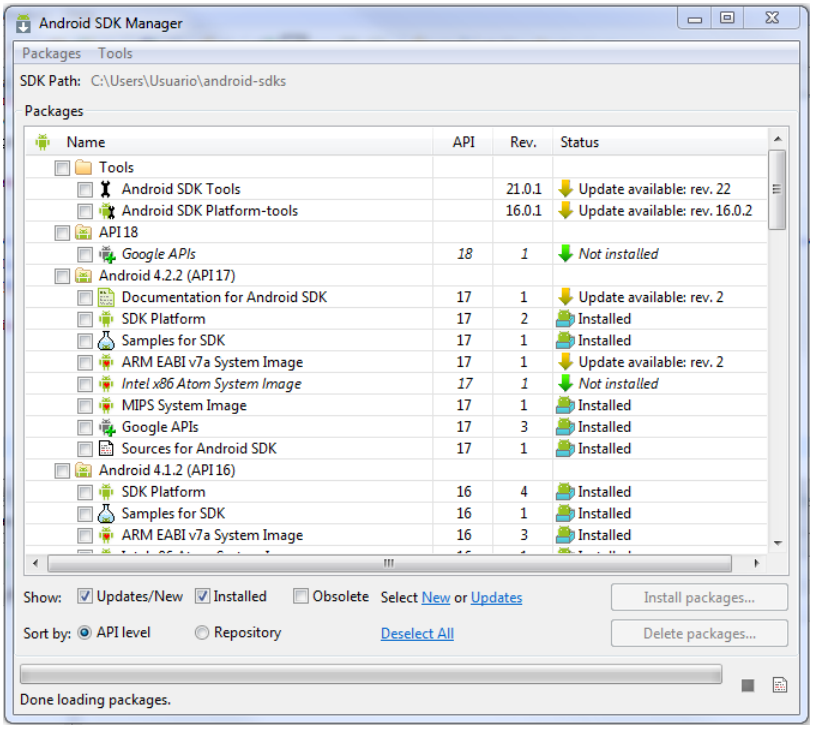

Figura B-1. Instalación de las diferentes API en eclipse

El SDK permite elegir en que versión de la plataforma se desea realizar la aplicación, para lo que se debe tener instaladas con anterioridad las APIs de la versión seleccionada. Las APIs se instalan desde la opción *Windows* → *Android SDK Manager*.

Plugin ADT

El ADT es un *plugin* para extender las capacidades del entorno de desarrollo Eclipse. Para ello, emplea las herramientas proporcionadas por el SDK agilizando la configuración de los proyectos y la creación de las interfaces de usuario.

El plugin ADT debe descargarse desde Eclipse, desde la opción *Help* → *Install New Software* e indicando la URL de descarga http://dl-ssl.google.com/android/eclipse/ en la que se buscarán todas las posibles actualizaciones, entre las que al menos ha de seleccionarse la de Android Developer Tools. Tras su instalación, se debe reiniciar Eclipse y configurar el ADT desde *Windows* → *Preferences* → *Android* indicando la ruta de ubicación del SDK.

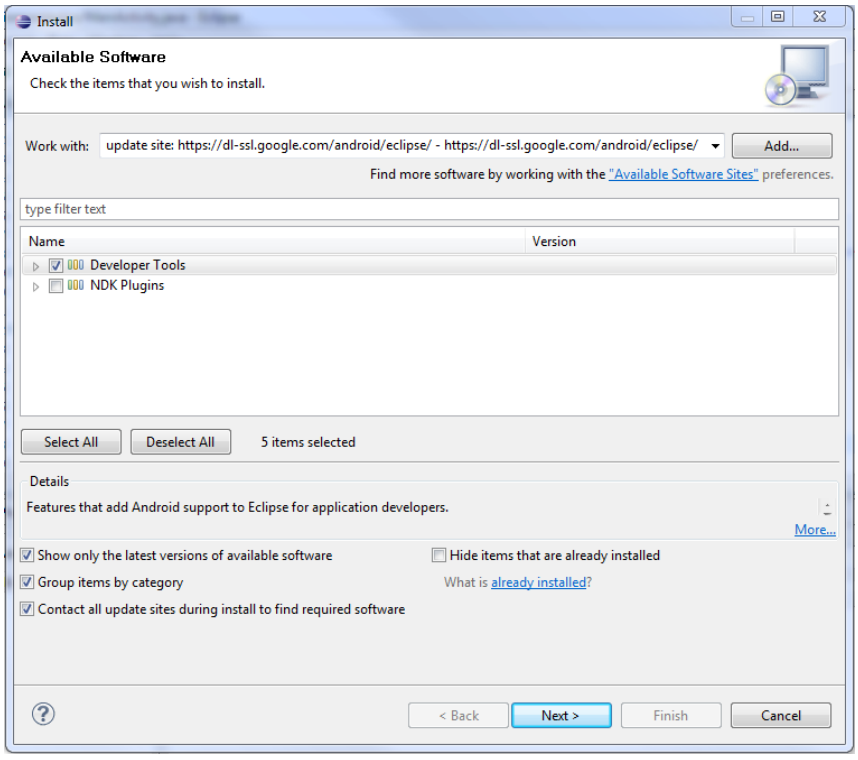

Figura B-2. Descarga del *plugin* ADT desde eclipse

AVD

El AVD o dispositivo virtual de Android permite emular desde Eclipse cualquier dispositivo con Android. De esta manera es posible probar las aplicaciones en una amplia variedad de *smartphones* con diferentes versiones de Android y de diferentes prestaciones hardware.

Se pueden crear varias AVD, cuya configuración se realiza desde Window → Android Virtual Device Manager. Los detalles sobre cada uno de los campos de configuración pueden consultarse en el apartado *Managing AVDs with AVD Manager* de [3].

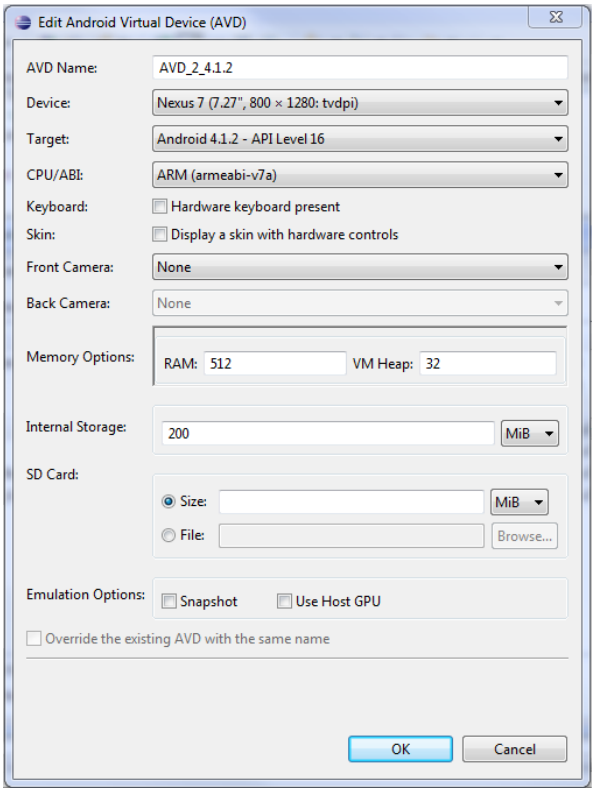

Figura B-3. Configuración de un AVD en eclipse

B.2.4. Apache Tomcat

Apache Tomcat es un servidor Web con soporte de servlets y JSPs. Puede ser descargado en un fichero comprimido desde la página oficial de Apache Tomcat [31]. La versión utilizada en la realización de este proyecto es la 6.0. Su instalación se produce al descomprimir el fichero en una ruta del sistema.

Para su configuración, se debe establecer la variable de entorno CATALINA_HOME para que apunte al directorio en el que se haya instalado Tomcat. Además se debe revisar que la variable JAVA_HOME apunte al directorio donde está instalado Java.

Apache Tomcat Eclipse Plugin

El Apache Tomcat Eclipse Plugin, como su propio nombre indica, es el plugin necesario para la ejecución del servidor en el entorno Eclipse.

El plugin se descarga desde Eclipse, desde la opción *Help* → *Install New Software* e indicando la URL de descarga http://download.eclipse.org/webtools/updates en la que se

buscarán todas actualizaciones posibles, entre las que se ha de seleccionar el paquete Servers para su instalación.al menos ha de seleccionarse la de Android Developer Tools. Tras ello, se ha de configurar el plugin desde la opción de eclipse *Windows* → *Preferences* → Server → *Runtime Environments* indicando la ruta de ubicación del Apache Tomcat.

Anexo C. Manual de usuario

C.1. Vulnerabilidades de seguridad

Para acceder a la recepción y gestión de vulnerabilidades de seguridad, se debe entrar en la aplicación y pulsar el botón *Vulnerabilidades* del menú de inicio (Figura C-1).

Figura C-1. Menú de inicio (I)

C.1.1. Activación/Desactivación de actualización de vulnerabilidades

La activación y desactivación de la actualización de vulnerabilidades se realiza desde el menú de *Vulnerabilidades* a través del botón *ON/OFF* (Figura C-2).

Al pulsar el botón, éste cambiar su estado de *OFF* a *ON* para lanzar la obtención de información sobre las posibles vulnerabilidades presentes en las aplicaciones instaladas en el dispositivo y de *ON* a *OFF* para dejar de recibirla. Con cada cambio se muestra un mensaje informativo para indicar si se ha iniciado o detenido la actualización de vulnerabilidades (Figura C-3).

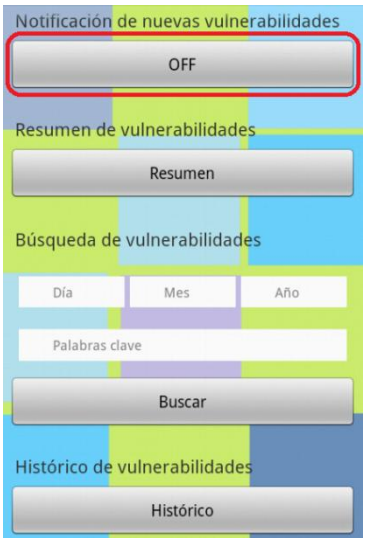

Figura C-2. Menú de Vulnerabilidades (I)

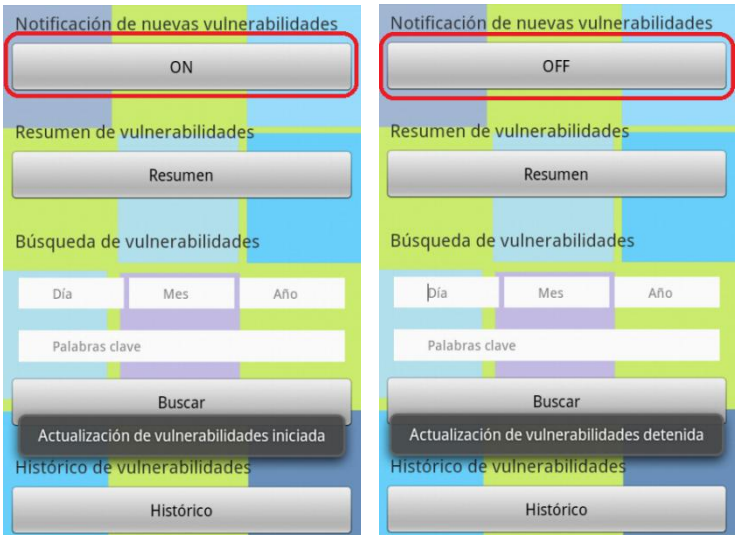

Figura 39. Menú de gestión de vulnerabilidades

Cuando se encuentran nuevas vulnerabilidades en el sistema, se muestra una notificación desplegable que indica el número de vulnerabilidades encontradas (Figura C-4). Al pulsar sobre la notificación, se accede la lista con las nuevas vulnerabilidades (Figura C-5), cada una de las cuales puede ser visualizada siguiendo los pasos explicados en el apartado C.1.3.

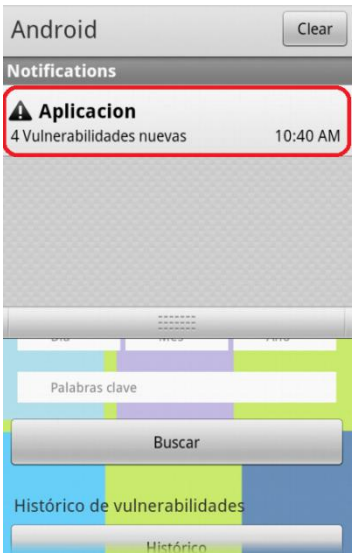

Figura 40. Notificación de vulnerabilidades encontradas

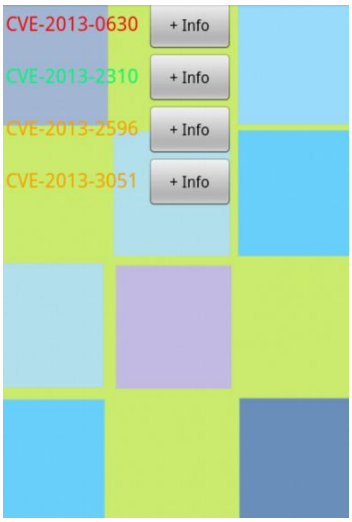

Figura C-5. Lista de vulnerabilidades (I)

C.1.2. Consulta de vulnerabilidades

- **Ver resumen y vulnerabilidades sin revisar**

  Para ver el resumen de vulnerabilidades de seguridad en el sistema, se debe pulsar el botón *Resumen* desde el menú de *Vulnerabilidades* (Figura C-6).

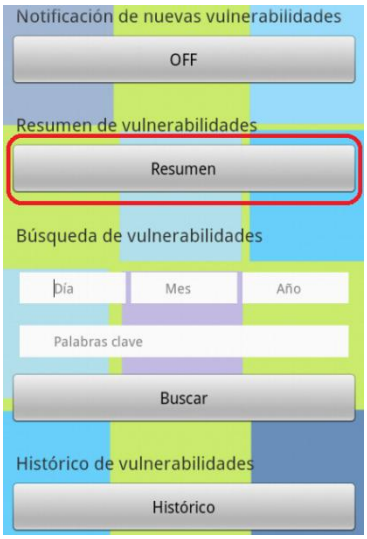

Figura C-6. Menú de Vulnerabilidades (II)

El resumen (Figura C-7) contiene una barra denominada *Revisadas* que indica visualmente el porcentaje de vulnerabilidades que han sido marcadas como revisadas respecto al total de vulnerabilidades registradas. En esta vista también se muestra en *Pendientes*, el número de vulnerabilidades por revisar frente al número total de vulnerabilidades almacenadas.

La lista de vulnerabilidades pendientes de ser revisadas, la cual sigue la misma estructura que en la Figura C-7, se obtiene pulsando el botón *Ver* del resumen. La visualización de estas vulnerabilidades se explica en el apartado C.1.3.

Figura C-7. Pantalla de resumen de vulnerabilidades

- **Búsqueda de vulnerabilidades**

  La realización de búsquedas de vulnerabilidades almacenadas en el sistema, en función de una determinada información, puede realizarse desde el menú de *Vulnerabilidades* (Figura C-8). Existen dos métodos diferentes de búsqueda:

mediante *Fecha*, por la que se obtienen las vulnerabilidades a partir de la fecha indicada (e incluyendo dicha fecha), y por *Palabras clave*, es decir, a través de una o más palabras relacionadas con la vulnerabilidad.

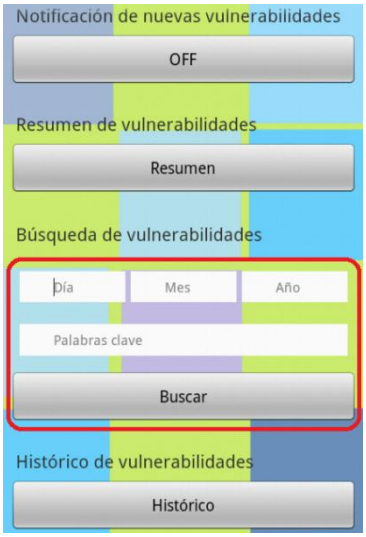

Figura C-8. Menú de Vulnerabilidades (III)

Los pasos para la búsqueda de vulnerabilidades por fecha son los siguientes:

1. Rellenar los campos *Día*, *Mes* y *Año* en la vista del menú *Vulnerabilidades*.
2. Pulsar el botón *Buscar*.

En el caso de querer realizar una búsqueda por palabra clave, los pasos a seguir son similares:

1. Escribir al menos una palabra en el campo *Palabras clave*. Si se escribe más de una palabra se deben separar por espacios en blanco.
2. Pulsar el botón *Buscar*.

También es posible hacer búsquedas combinando ambos criterios.

En caso de que los campos a rellenar se encuentren incompletos y no sea posible realizar una búsqueda, se muestra un mensaje avisando de ello (Figura C-9).

Tras pulsar el botón Buscar se obtiene una lista, como la de la Figura 5, con las vulnerabilidades que cumplen con los criterios de búsqueda establecidos. La visualización de éstas se explica en el apartado C.1.3.

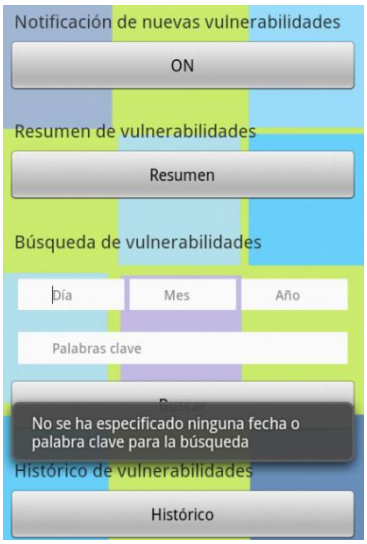

Figura C-9. Mensaje de aviso de campos incompletos en la búsqueda

- **Histórico de vulnerabilidades**

  La consulta de todas las vulnerabilidades registradas en el sistema, se lleva a cabo a través del botón *Historico* del menú de *Vulnerabilidades* (Figura C-10). Al pulsarlo, se accede a la vista con la lista de vulnerabilidades, la cual sigue la estructura de la Figura 5. Los datos de cada vulnerabilidad pueden visualizarse siguiendo los pasos explicados en el apartado C.1.3.

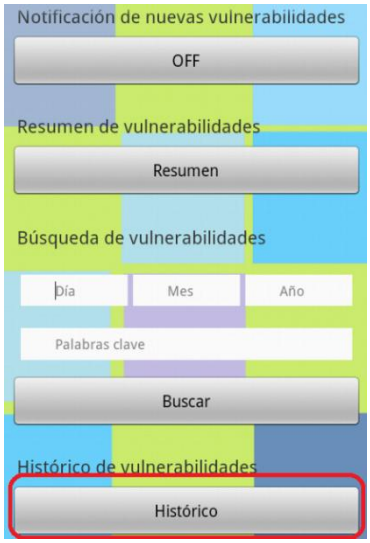

Figura C-10. Menú de Vulnerabilidades (IV)

C.1.3. Visualización de la información de una vulnerabilidad

Desde la vista con la lista de vulnerabilidades, a la que se puede acceder a través de cualquiera de las formas presentadas anteriormente (notificación, resumen, búsqueda e histórico), se puede ver la información de cada vulnerabilidad siguiendo los siguientes pasos:

1. Pulsar el botón *+Info* correspondiente a la vulnerabilidad que se desea visualizar (Figura C-11).

   El color del identificador de cada vulnerabilidad está relacionado con su relevancia en la seguridad del sistema, desde verde, el más permisivo hasta rojo, el más severo.

Figura C-11. Lista de vulnerabilidades (II)

2. La información de la vulnerabilidad queda presentada como en la Figura C-12. En la mayoría de casos es necesario realizar *scroll*[12] para poder ver todos sus datos.

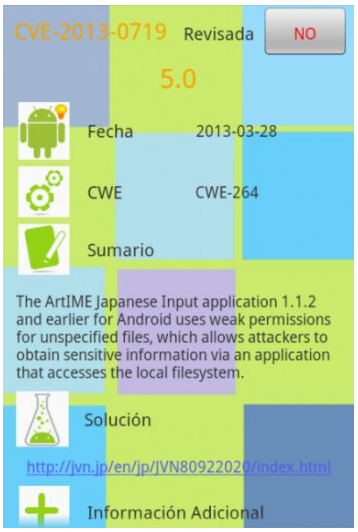

Figura 41. Pantalla de visualización de una vulnerabilidad (I)

C.1.4. Revisión de una vulnerabilidad

Para marcar una vulnerabilidad como revisada, se debe partir de la pantalla de visualización de la misma.

[12] *Scroll:* desplazamiento en una vista para su visualización completa.

1. Acceder a la página Web del fabricante a través del hiperenlace indicado en *Solución* (Figura C-13).

   Este hiperenlace conduce a la página Web con la solución propuesta por el fabricante de la aplicación con la vulnerabilidad señalada.

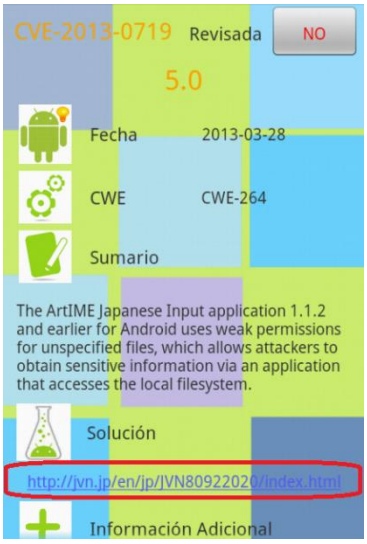

Figura 42. Pantalla de visualización de una vulnerabilidad (II)

2. Consultar la solución indicada en la página Web y aplicarla en el dispositivo (Figura C-14).

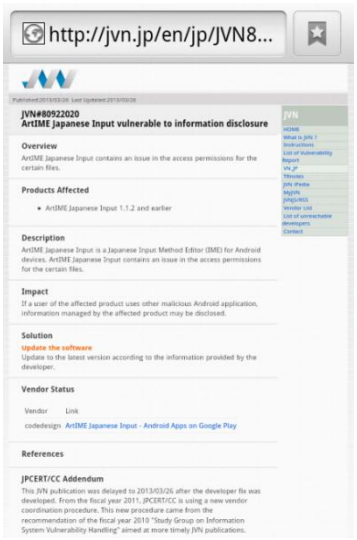

Figura C-14. Página Web del fabricante del producto con vulnerabilidad

3. Volver a la pantalla de visualización de la vulnerabilidad y cambiar su estado *Revisada* de *NO* a *SI* pulsando el botón indicado para ello (Figura C-15).

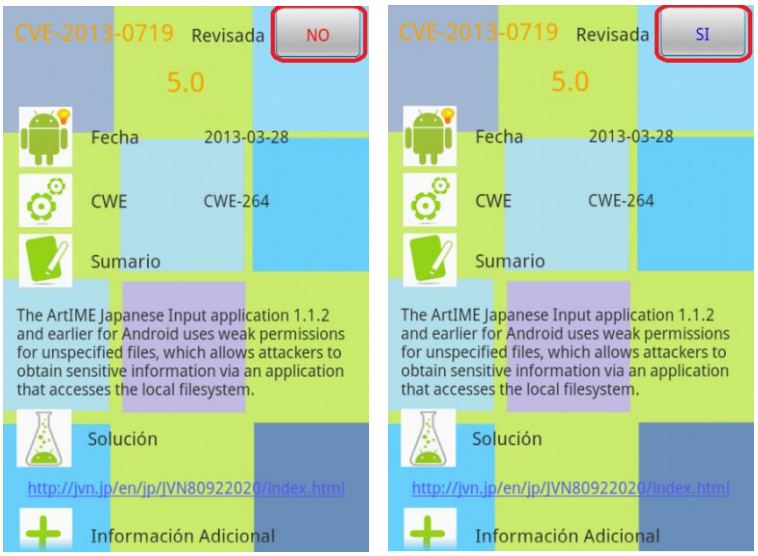

Figura 43. Pantalla de visualización de una vulnerabilidad (III)

C.2. Contextos de seguridad

Para acceder a la parte encargada de la aplicación y gestión de los contextos de seguridad, se debe abrir la aplicación y pulsar el botón *Seguridad por contexto* del menú de inicio (Figura C-16).

Figura C-16. Menú de inicio (II)

C.2.1. Creación de un contexto

Para poder detectar contextos, se hace necesario, primeramente, la creación y definición de los mismos. Para la creación de un contexto es preciso estar situado en el lugar que se desea señalar como contexto. Una vez dentro del menú de *Seguridad por contexto*, los pasos a seguir para la creación de un contexto son los siguientes:

1. Pulsar el botón *Nuevo* del menú (Figura C-17).

2. En la vista principal (Figura C-18) se muestran los campos a rellenar para la creación del contexto: nombre del contexto, radio, hora de inicio y hora de fin. La hora debe estar comprendida entre las 00 y las 23 y los minutos entre 00 y 59. Si alguno de estos campos se rellena de forma incorrecta, la aplicación avisa con su mensaje de información correspondiente.

   Para obtener los datos de localización del contexto que se está creando, se ha de pulsar el botón *Localización actual*, encargado de mostrar en los campos correspondientes, las coordenadas de latitud y longitud en las que se encuentra el dispositivo.

Figura C-17. Menú de Seguridad por contexto (I)

Figura C-18. Pantalla para la creación de un contexto

Para la elección del tipo de seguridad del contexto, se dispone de un menú de selección para entre dos opciones, *Alta* y *Baja* (Figura C-19).

3. Una vez rellenos todos los campos, pulsar el botón *Guardar*, al final de la pantalla, para grabar el contexto.

   El campo *radio*, se refiere al radio del contexto en metros.

   En caso de que alguno de los datos del contexto se encuentre sin completar y se pulse el botón *Guardar*, la aplicación muestra un mensaje de aviso señalando la necesidad de rellenar todos los campos para que el contexto pueda ser creado (Figura C-20).

Figura C-19. Selección del tipo de seguridad

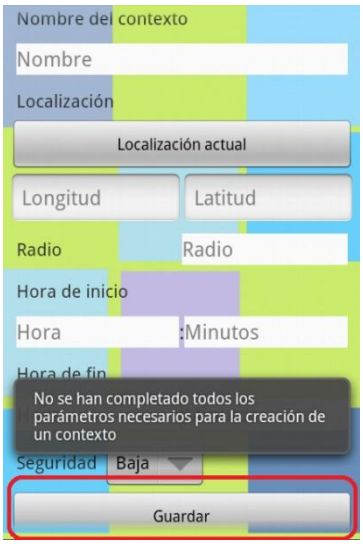

Figura C-20. Mensaje de aviso de campos incompletos en el contexto

C.2.2. Visualización de un contexto

Desde el menú de *Seguridad por contexto*, es posible visualizar los contextos que hayan sido previamente creados en el sistema. Los pasos para la consulta de los datos de un contexto son los explicados a continuación.

1. Pulsar el botón *Contextos* del menú (Figura C-21).
2. Una vez listados los contextos almacenados en el sistema, pulsar el botón *Ver* del contexto que queramos consultar (Figura C-22).

La Figura C-23 muestra el aspecto de un contexto al ser consultado por el usuario. Con la visualización un contexto, a su vez, se ofrecen las opciones para su modificación o eliminación, las cuales se explican más abajo.

Figura C-21. Menú de Seguridad por contexto (II)

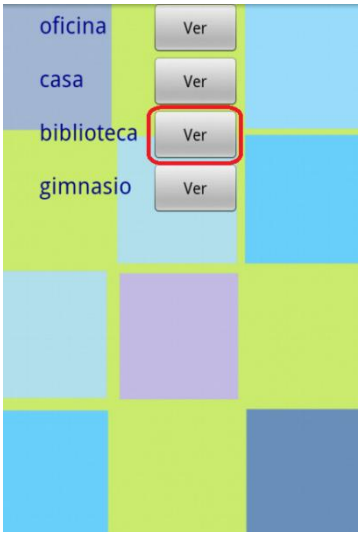

Figura 44. Listado de contextos en el sistema

Figura C-23. Pantalla de visualización de un contexto (I)

C.2.3. Modificación de un contexto

La modificación de un contexto se realiza desde la pantalla de visualización del mismo. Para la realización de cambios, se debe proceder de la siguiente manera.

1. Pulsar el botón *Modificar* de la pantalla de visualización del contexto a modificar (Figura C-24).

Figura C-24. Pantalla de visualización de un contexto (II)

2. Rellenar los campos que se deseen cambiar en la pantalla para la modificación del contexto (Figura C-25).

   En cada uno de ellos, se muestran los datos actuales del contexto a modo de referencia. Los campos que queden sin rellenar mantendrán su valor anterior.

Figura C-25. Pantalla para la modificación de un contexto

3. Pulsar el botón *Guardar Cambios* para grabar los cambios realizados en el contexto.

C.2.4. Eliminación de un contexto

Para la eliminación de un contexto, se debe partir de su pantalla de visualización. Desde ella, se han de seguir los siguientes pasos.

1. Pulsar el botón *Eliminar* de la pantalla de visualización del contexto que se va a proceder a eliminar (Figura C-26).

Figura C-26. Pantalla de visualización de un contexto (III)

2. Confirmar la acción de borrado pulsando el botón *Confirmar* del mensaje de aviso que lanza la aplicación al presionar el botón *Eliminar* en el paso anterior o por el contrario, volver atrás mediante la opción *Cancelar* (Figura C-27).

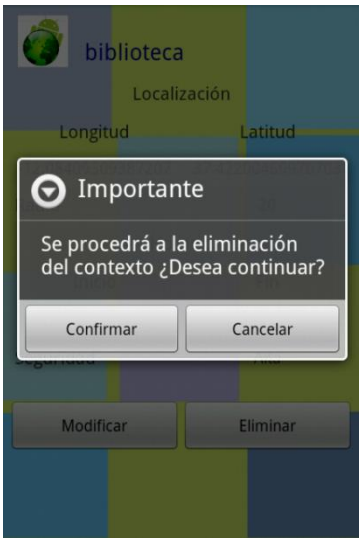

Figura C-2745. Mensaje para la confirmación de la eliminación de un contexto

Una vez se confirme la eliminación del contexto no se podrá recuperar por lo que para volver a disponer de un contexto igual, éste deberá ser vuelto a crear en el sistema.

C.2.5. Activación/Desactivación del uso de contextos

La activación y desactivación del uso de contextos de seguridad se puede realizar desde el menú de *Seguridad por contexto* a través del botón *ON/OFF* indicado para ello (Figura C-28).

Al ser pulsado, éste cambiar su estado de *OFF* a *ON* para comenzar el uso de contextos y de *ON* a *OFF* para dejar de utilizarlos. Asimismo, con cada cambio se muestra un mensaje informativo sobre la iniciación o detección del uso de contextos (Figura C-29).

Figura C-28. Menú de Seguridad por contexto (III)

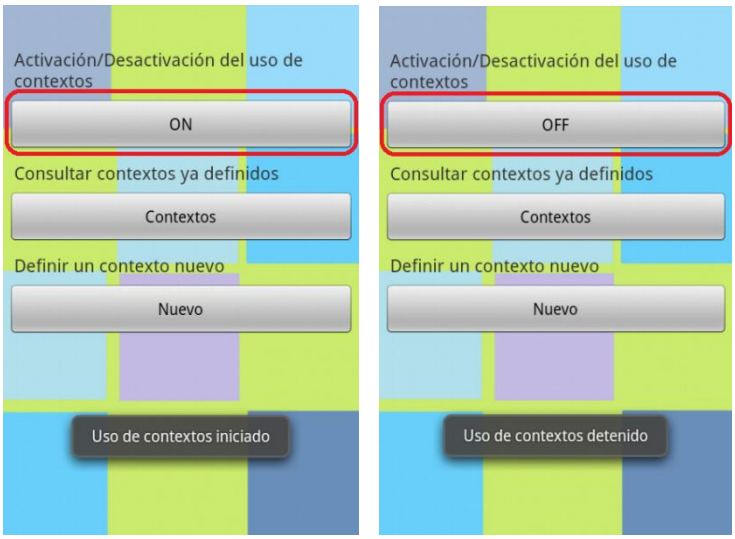

Figura C-2946. Activación y desactivación del uso de contextos

Anexo D. NIDS

D.1. Introducción

Un NIDS [32] es un sistema de seguridad para la detección de intrusos (IDS) que analiza los datos transportados en un segmento de red y produce alertas cuando detecta anomalías.

Su funcionamiento está basado en la monitorización del tráfico que circula por la red con el fin de localizar accesos no autorizados o la utilización incorrecta de datos. La detección del tráfico infectado se realiza utilizando un conjunto de reglas definidas que permiten establecer comparaciones para, en caso de incongruencias, la generación de las alertas. Habitualmente, estas alertas se recogen como *logs* en ficheros de texto o bases de datos para ser utilizadas posteriormente.

El *software* necesario para su ejecución [33] puede estar instalado en el mismo dispositivo cuyo tráfico queremos analizar o bien en un servidor independiente, trabajando en modo promiscuo y capturando todo el tráfico del segmento de red para su posterior filtrado. Cabe señalar que, en el primer caso, sólo se monitoriza el tráfico de una sola máquina, en la que el NIDS está instalado, mientras que en el segundo caso es posible analizar el tráfico de un conjunto de máquinas.

Existen diversas propuestas de NIDS para Android, aunque sólo algunas de ellas han llegado a ser implementadas hasta la fecha, debido a las dificultades presentadas en la monitorización en tiempo real de la información utilizada por las aplicaciones de terceros en los *smartphones*, tal y como la limitación de recursos.

Asimismo, a pesar de ser posible la utilización de un análisis estático para la detección de *software* infectado en dispositivos Android, opción principalmente propulsada por las compañías de antivirus, nuevamente la limitación de recursos en estos dispositivos hace que lo más conveniente sea una detección dinámica basada en el análisis de comportamiento. Algunos de estos NIDS dinámicos son Crowdroid [19] y Paranoid Android [20], que desarrollan su funcionamiento fuera del dispositivo liberando de carga al mismo, mientras

que otros como Taintdroid [21] y Mockdroid [22] realizan toda su actuación en él. Estos NIDS se presentan a continuación y sus características quedan recogidas en la tabla comparativa D-1.

NID	Modo de actuación	Características	Lugar de actuación	Disponibilidad
Crowdroid	Monitorización de las llamadas al sistema	Depende de la cantidad de información disponible	Servidor remoto	No disponible
Paranoid Android	Simulación en entorno virtual	Servicio a través de la nube. Escalable	Servidor remoto	No disponible
TaintDroid	Tintado de la información relevante	Granularidad en el etiquetado. Permite el rastreo simultáneo	Dentro del dispositivo	Código fuente disponible
Mockdroid	Limitación de permisos	Granularidad en los permisos otorgados	Dentro del dispositivo	Código fuente disponible

Tabla 88. Resumen y comparación de NIDS para Android

D.2. NIDS para Android

D.2.1. Crowdroid

Crowdroid [19] es un sistema de detección de malware que está basado en el análisis de los patrones de comportamiento de las aplicaciones instaladas en los dispositivos móviles Android. Concretamente, en las llamadas al *kernel* de Linux del sistema.

La arquitectura de este *framework* está formada por tres bloques. La adquisición de datos, su manipulación y finalmente el análisis y detección del *malware*.

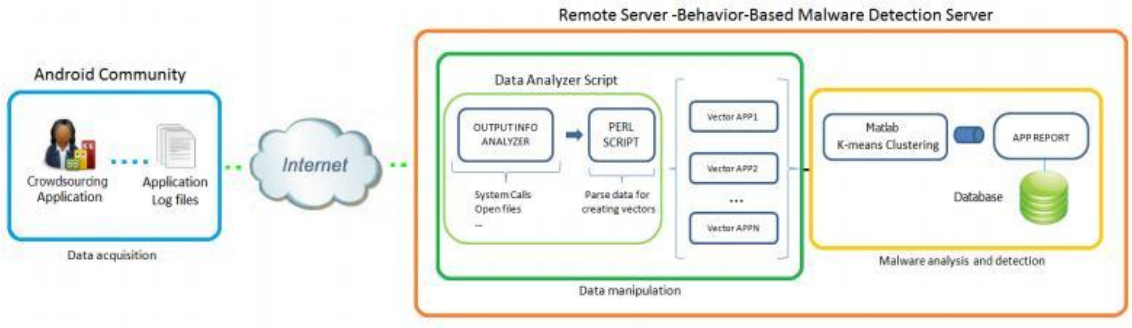

Figura 47. Bloques de la arquitectura de Crowdroid [19]

Su mecanismo de funcionamiento consta de un cliente ligero para la monitorización de las llamadas al sistema y posterior envío de la información recopilada a un servidor remoto centralizado. Este servidor es el encargado de manipular los datos recibidos y de la creación

de un vector con las llamadas al sistema por cada una de las aplicaciones del usuario. Las más de 250 diferentes llamadas al sistema se identifican con un número único, de forma que las posiciones del vector de llamadas al sistema se corresponden con cada una de las peticiones existentes y cada elemento del mismo contiene una cifra con el número de llamadas al sistema realizadas por una determinada aplicación. Para el análisis de los datos, se comparan los vectores de llamadas a sistema de aplicaciones con igual nombre y versión, obteniéndose una matriz de distancias euclídeas entre ellos de forma que los valores cercanos a 0 indican vectores similares y valores más altos señalan diferencias entre el comportamiento de las aplicaciones comparadas. Por último, cada uno de los vectores es clasificado siguiendo un algoritmo de agrupamiento k-medias, por el que se establecen dos centroides, uno para las aplicaciones malignas y otro para las benignas de forma que, en función de la cercanía a los mismos, se distinguen las aplicaciones infectadas de las que no lo están.

Tal y como puede deducirse de su funcionamiento, la efectividad del proceso de detección de intrusos de Crowdroid reside en la cantidad de información de la que se disponga, es decir, del número de vectores de llamadas al sistema recopilados. Es por esto que la precisión a la herramienta depende de la cantidad de dispositivos que hagan uso de ella lo que con una amplia recopilación de datos se convierte en una herramienta potente. Sin embargo, se trata de un NIDS cuyo cliente no se encuentra disponible para ser instalado en dispositivos Android, a pesar de que en [19] se haga referencia a que puede ser descargado desde el Market de Google, por lo que a pesar de tratarse de un NIDS práctico y de funcionamiento sencillo, no se presenta como una opción válida para su uso.

D.2.2. Paranoid Android

Paranoid Android [20] es un modelo de seguridad implementado en servidores remotos que tiene como objetivo la detección de amenazas en *smartphones* con sistema operativo Android, incluyendo nuevos ataques, mediante la simulación de los dispositivos en un entorno virtual.

Su arquitectura se compone por dos grandes bloques, el de recopilación de datos y el de su simulación.

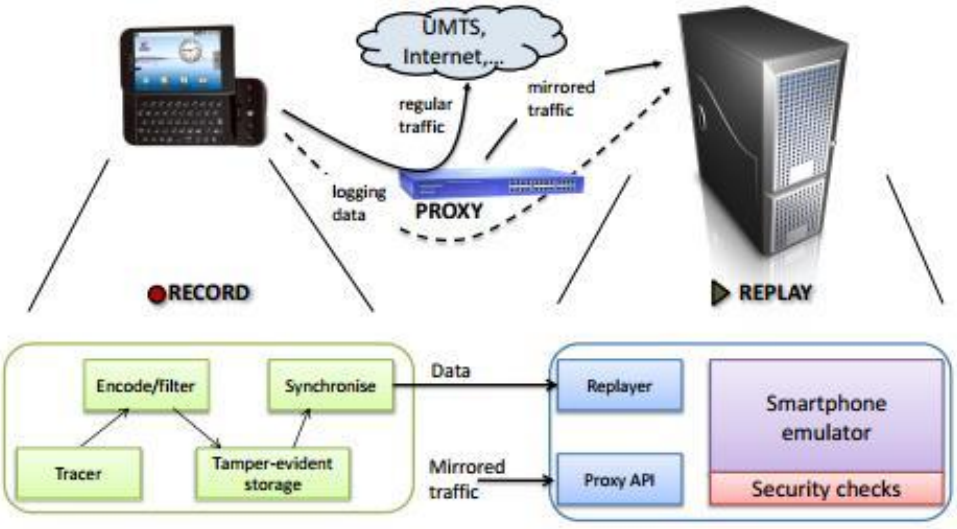

Figura 48. Bloques de la arquitectura de Paranoid Android [20]

Para su funcionamiento, consta de un programa conocido como *tracer* que, instalado en el dispositivo, se encarga de registrar toda la información necesaria para su posterior ejecución. Esta información está formada en su mayoría por los datos transferidos desde el *kernel* del sistema hasta el espacio de usuario a través de las llamadas al sistema y de otros mecanismos como señales. La traza resultante es enviada a la nube a través de un canal cifrado y una vez en el servidor remoto, se simula la ejecución de la misma mediante un emulador o *replayer*, con el objetivo de realizar chequeos de seguridad en tiempo real que permitan detectar comportamientos inadecuados sin que ello suponga un consumo extra de recursos y consumo de batería en el dispositivo. Estos chequeos pueden basarse en diferentes mecanismos de detección como la utilización de escáneres de virus o análisis de tintado. Finalmente, se informa a los dispositivos acerca de los resultados obtenidos. Con el fin de que no sea necesaria la retransmisión de las trazas de información desde el *smartphone* hacia el servidor, se emplea un *proxy* en la conexión hacia internet, que permite el almacenamiento temporal del tráfico entrante.

Como puede observarse, una característica importante de este NIDS es que se trata de un servicio de seguridad que puede ofrecerse a través de la nube con una arquitectura escalable en la que, en principio, no existe límite en el número de técnicas de detección de ataques que pueden ser utilizadas en paralelo. Asimismo, con el uso de un servidor remoto se eliminan las limitaciones de procesado impuestas por los dispositivos móviles. No obstante, se trata nuevamente de un prototipo sin implementación real por lo que por el momento no es posible su utilización como complemento de seguridad.

D.2.3. TaintDroid

TaintDroid [21] es una extensión de seguridad para Android creada por investigadores de la Universidad Estatal de Pennsylvania, la Universidad de Duke y los laboratorios de Intel. Desarrolla su función a partir de técnicas de análisis de tintado, las cuales permiten el seguimiento de la información privada en las aplicaciones móviles de terceros sospechosas de estar infectadas.

Taintdroid trabaja en todos los niveles de la arquitectura de Android: a nivel de aplicación, espacio de usuario y *kernel*.

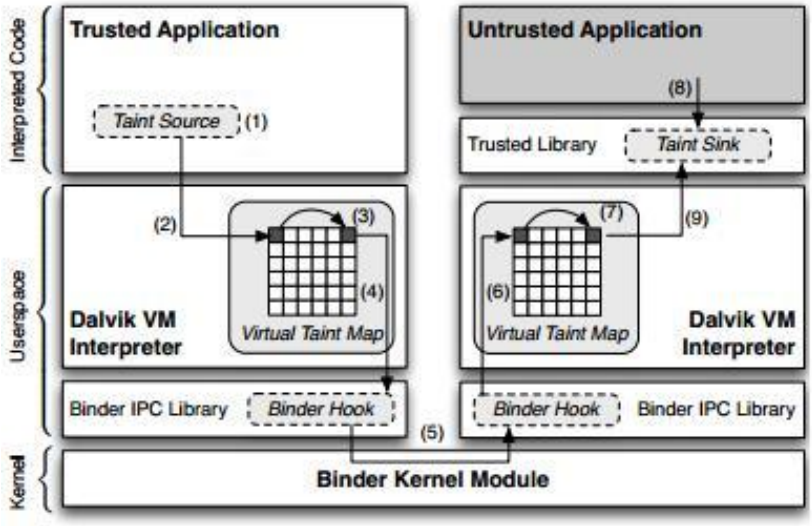

Figura 49. Arquitectura de Taintdroid en Android [21]

Para poder realizar el análisis de las aplicaciones, TaintDroid parte de una aplicación fiable en la que se lleva a cabo el etiquetado de la información privada. La información es identificada en la *taint source*, en la que se asignan las etiquetas, conocidas como *taint markings*. Estas etiquetas son asignadas teniendo en cuenta la propagación de dicha información por los elementos que forman parte de la aplicación, estableciéndose niveles de propagación de tintado. Tras ello, las *taint markings* son almacenadas en un mapa virtual de etiquetas o *taint map* a través de la máquina virtual Dalvik. Finalmente, antes de que la información relevante abandone el dispositivo, es identificada en el *taint sink* mediante *taint tags*, generalmente en la interfaz de red. Toda esta información es propagada de forma transparente a través del *kernel* del sistema y recibida en la aplicación sospechosa de estar infectada. Cuando la aplicación sospechosa requiere de datos identificados dentro del *taint sink*, se extrae la información relacionada almacenada en el *taint map* y se reporta el evento. Dentro de esta información se encuentran los propios datos enviados, la aplicación

encargada de su transmisión y su destino. De esta manera los usuarios pueden conocer qué aplicaciones parecen tener un comportamiento inadecuado.

Taintdroid permite el rastreo simultáneo de múltiples fuentes de datos relevantes y su granularidad en el etiquetado hace que la detección sea eficiente. Además, cuenta con una página de desarrollo [34] desde la que se puede descargar el código fuente. Sin embargo, no se trata de un módulo de fácil instalación como puede desprenderse de las instrucciones disponibles para su puesta en marcha en la versión Android 4.1 también disponibles en [34].

D.2.4. Mockdroid

Mockdroid [22] es una versión modificada del sistema operativo Android desarrollada por la Universidad de Cambridge que permite limitar el acceso de una aplicación a un recurso en tiempo de ejecución, para lo que lo muestra como inaccesible cuando una aplicación sin permisos solicita el acceso al mismo. Asimismo, permite a los usuarios la prohibición de acceso directa a determinados recursos.

Antes de la descarga e instalación de una aplicación Android desde el Market, se informa al usuario sobre los permisos que deben ser aceptados para ello, y en caso de ser aceptados, son almacenados en memoria. Después, en la ejecución de las aplicaciones, con cada llamada a la API que requiera de un permiso, se comprueba si se dispone de dicho permiso mediante la consulta de la información almacenada anteriormente en la memoria y si no aparecen como permiso aceptado, se devuelve una excepción. La modificación introducida por Mockdroid reside en que primero se comprueba si el permiso fue requerido antes de la instalación y de no haberlo sido, se devuelve una excepción de la misma manera que en el estándar, pero en el caso de haberlo sido, también se comprueba si el usuario decidió bloquear ciertos recursos aunque fuesen asignados en tiempo de instalación, devolviendo una respuesta válida pero incorrecta, cuyo ejemplo más habitual es responder que no se dispone de los datos requeridos en ese momento por falta de conexión.

Las aplicaciones cuyas peticiones de acceso fueron denegadas continúan trabajando en Mockdroid debido a que en su mayoría están diseñadas para tolerar pequeños fallos tales como la ausencia de conexión o señal GPS, aunque probablemente lo hagan con funcionabilidad reducida. Además, con su utilización se asegura la protección frente a conexiones a Internet o escritura de mensajes indeseadas y el control de los datos de identificación del dispositivo o de localización. Mockdroid puede ser descargado desde su página de desarrollo [35] aunque solo está disponible para HTC Nexus One.

Anexo E. Glosario de términos

API Application Programming Interface. Es un conjunto de métodos y procedimientos recogidos en una biblioteca y que definen cómo invocar desde un programa cada uno de los servicios que éstos prestan.

AVD Android Virtual Device. Se trata de la configuración de un emulador que permite modelar un dispositivo Android real mediante la definición de su hardware y software en el emulador de Android.

BBDD Base de datos.

CGI Common Gateway Interface. Es la interfaz entre los servidores Web y las aplicaciones que se ejecutan en el servidor

Cookie Una cookie es un fichero enviado a un navegador por medio de un servidor Web para registrar las actividades de un usuario en un sitio

DOM Document Object Model. Método para la lectura y tratamiento de ficheros XML que devuelve el contenido resultado de la lectura completa del documento en forma de una estructura de tipo árbol.

GPS Global Positioning System. Es un sistema global de navegación por satélite que permite determinar la posición de un objeto en cualquier punto de la Tierra con una precisión de hasta centímetros, siendo lo habitual unos pocos metros de precisión.

GPX GPS eXchange Format. Formato ligero de datos XML para el intercambio de información GPS (puntos, recorridos y rutas) entre aplicaciones y servicios Web en Internet.

HTML HyperText Markup Language. Es un lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto que es el formato estándar de las páginas Web.

HTTP Hypertext Transfer Protocol. Es el protocolo utilizado para transferir ficheros de hipertexto, es decir, páginas Web por Internet.

IDE Integrated Development Environment. Es una aplicación formada por un conjunto de herramientas de programación. Suele consistir en un editor de código, un compilador, un *debugger* y un constructor de interfaz gráfica.

Javascript Es un lenguaje interpretado con una sintaxis parecida a la del lenguaje Java, utilizado principalmente en páginas Web para realizar tareas y operaciones en el marco de la aplicación cliente.

JSP JavaServer Pages. Tecnología para la creación de contenido Web dinámico utilizando el lenguaje de programación Java. Para su uso se requiere de un servidor Web compatible con contenedores Servlet como Apache Tomcat o Jetty.

JRE Java Runtime Environment. Es un conjunto de utilidades que permite la ejecución de programas Java. Se compone de un conjunto de bibliotecas Java y otros componentes necesarios para ejecutar *applets* y aplicaciones escritas en Java.

JVM Java virtual Machine. Máquina virtual capaz de interpretar y ejecutar instrucciones generadas por el compilador de Java, es decir, se trata de un puente entre el resultado de la compilación y el sistema sobre el que se ejecuta la aplicación.

Kernel Es el núcleo o parte fundamental de un sistema operativo. Este software se encarga de gestionar los recursos del sistema.

KML Keyhole Markup Language. Es un lenguaje de marcado basado en XML para representar datos geográficos en tres dimensiones. Fue desarrollado para ser manejado con Keyhole LT, precursor de Google Earth.

LRU Least Recently Used. Algoritmo de reemplazo por el que se cae de la lista el elemento que más tiempo hace que no ha sido utilizado.

NDK Native Development Kit. Es un conjunto de herramientas que permite incorporar componentes que hacen uso de código nativo con lenguajes como C y C++.

NIST U.S. National Institute of Standards and Technology. Es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos cuya misión es promover la innovación y la competencia industrial en los Estados Unidos mediante avances en metrología, normas y tecnología.

Proxy Un servidor proxy es un equipo intermedio entre el sistema final e Internet. Se utiliza para registrar el uso de Internet y bloquear el acceso a algunas páginas Web.

Phishing Es un ataque informático consistente en el envío masivo de mensajes que, aparentemente proveniente de fuentes fiables, intenta obtener datos confidenciales proporcionados por el usuario.

Plugin Es un módulo que se anexa opcionalmente a un programa para proporcionarle nuevas características o funcionalidades al mismo.

SAX Simple API for XML. Método para la lectura y tratamiento de ficheros XML que a medida que lee secuencialmente el documento XML va generando eventos con la información de cada elemento leído.

SCAP Security Content Automation Protocol. Se trata de un conjunto de especificaciones del NIST para mostrar y tratar la información relacionada con la seguridad, como mediciones o vulnerabilidades, de una forma estandarizada.

SDK Software Development Kit. Es un conjunto de herramientas y programas de desarrollo que permiten al programador crear aplicaciones para un determinado paquete de software o un sistema operativo.

URI Uniform Resource Identifier. Es una cadena que funciona como identificador global de un recurso en la Web tales como documentos, archivos descargables, servicios, buzones de correo y otros. Está compuesta por el protocolo, la dirección IP o nombre del terminal, la ruta de directorios opcionalmente, la consulta (usualmente pares clave = valor) y el nombre del archivo.

URL Uniform Resource Locator. Se trata de una secuencia de caracteres de acuerdo a un formato estándar que se usa para identificar una ubicación en la Web. Una URL está formada por el protocolo, la dirección IP o nombre del terminal, la ruta de directorios opcionalmente y el nombre del archivo.

XML eXtensible Markup Language. Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium y que permite definir lenguajes de acuerdo a necesidades.

Bibliografía

[1] (U//FOUO) DHS-FBI Bulletin: Threats to Mobile Devices Using the Android Operating System. Disponible en <http://info.publicintelligence.net/DHS-FBI-AndroidThreats.pdf>. Consultado en octubre 2013.

[2] Open handset Alliance, Android. Disponible en <http://www.openhandsetalliance.com>. Consultado en marzo 2013.

[3] Desarrollo en Android. Disponible en <http://developer.android.com>. Última consulta en agosto 2013.

[4] Gartner research. Disponible en <http://www.gartner.com>. Consultado en marzo 2013

[5] Tomás Gironés, J. *El gran libro de Android*. Barcelona: Marcombo S.A., 2013. ISBN: 978-84-267-1976-8.

[6] Seguridad en Android. Disponible en <http://www.ibm.com/developerworks/ssa/library/x-androidsecurity>. Consultado en marzo 2013.

[7] M. Conti, V.T.N. Nguyen, and B. Crispo. CRePe: Context-Related Policy Enforcement for Android. Florida, USA, 2010.

[8] Vanegas, C.S. *Los servlets como puente de comunicación cliente-servidor empleando java*. 2006.

[9] Alonso Blázquez, F., Serrano Bárcena, N. y Calzada Mínguez, S. *Informática II*. Capítulo 9: Servlets. Universidad de Navarra, 2007.

[10] Muñoz Organero, M. y Fernández, N. *Servlets (I).* Universidad Carlos III Madrid. Disponible en <http://www.it.uc3m.es/mario/si/servlets-1.pdf>. Consultado en febrero 2013.

[11] Apache software foundation. Disponible en <http://apachefoundation.wikispaces.com>. Consultado en febrero 2013.

[12] NVD. Disponible en <http://nvd.nist.gov>. Consultado en abril 2013.

[13] CPE. Disponible en <http://cpe.mitre.org>. Consultado en abril 2013.

[14] CVE. Disponible en <http://cve.mitre.org>. Consultado en abril 2013.

[15] CWE. Disponible en <http://cwe.mitre.org>. Consultado en abril 2013.

[16] CWE, NIST. Disponible en <http://nvd.nist.gov/cwe.cfm>. Consultado en abril 2013.

[17] CVSS, NIST. Disponible en <http://nvd.nist.gov/cvss.cfm>. Consultado en abril 2013.

[18] CVSS guide. Disponible en <http://www.first.org/cvss/cvss-guide.html>. Consultado en abril 2013.

[19] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. *Crowdroid: behaviour-based malware detection system for Android*. Chicago, USA, 2011.

[20] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: versatile protection for smartphones". Texas, USA, 2010.

[21] W. Enck, P. Gilbert, and B.G. Chun, "Taintdroid: An information-Flow Tracking Systems for Realtime Privacy Monitoring on Smartphones". Vancuver, Canada, 2010.

[22] A.R. Beresford, A. Rice and N. Skehin, "Mockdroid: trading privacy for application functionality on smartphones". Phoenix, USA, 2011.

[23] AroundMe. Disponible en <https://play.google.com/store/apps/details?id=com.tweakersoft.aroundme&hl=es>. Consultado en julio 2013.

[24] Tripwolf. Disponible en <http://www.tripwolf.com/es/page/android>. Consultado en julio 2013.

[25] Descarga del fichero *nvdcve-2.0-2013.xml*. Disponible en <http://nvd.nist.gov/download.cfm>.

[26] Localización geográfica en Android. Disponible en <http://www.sgoliver.net/blog/?p=1932>. Consultado en julio 2013.

[27] A. Gupta, M. Miettinen and N. Asokan. *Intuitive security policy configuration in mobile devices using context profiling*. Purdue University, 2011.

[28] Geofencing. Disponible en <http://developer.android.com/training/location/geofencing.html>. Consultado en julio 2013.

[29] Descarga de Java. Disponible en <http://www.java.com>.

[30] Descarga de Eclipse. Disponible en <http://www.eclipse.org>.

[31] Descarga de Tomcat. Disponible en <http://tomcat.apache.org>.

[32] A. Rodrigo. *NIDS (Network Intrusion Detection Systems)*. Universidad Politécnica de Cataluña. Disponible en <http://studies.ac.upc.edu/FIB/CASO/seminaris/2q0304/M6.pdf>. Consultado en febrero 2013.

[33] Network Intrusion Detection Systems. Disponible en <http://www.linuxsecurity.com/resource_files/intrusion_detection/network-intrusion-detection.html>. Consultado en diciembre 2012.

[34] Realtime Privacy Monitoring on Smartphones, Taintdroid. Disponible en <http://appanalysis.org>. Consultado en diciembre 2012.

[35] University of Cambridge research, MockDroid. Disponible en <http://www.cl.cam.ac.uk/research/dtg/android/mock>. Consultado en diciembre 2012.